TECHNISCHE
UNIVERSITÄT
DARMSTADT

Automatisierte Analyse integrierter Software-Produktlinien-Spezifikationen

DEM FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

MARKUS TOBIAS WECKESSER

REFERENT: PROF. DR. RER. NAT. ANDREAS SCHÜRR
KORREFERENT: PROF. DR. PHIL. NAT. CHRISTIAN BECKER

TAG DER EINREICHUNG: 12.03.2019
TAG DER DISPUTATION: 28.06.2019

D17
DARMSTADT 2019

Die Arbeit von Markus Tobias Weckesser wurde unterstützt durch den Sonderforschungsbereich (SFB) 1053 *Multi-Mechanismen-Adaptation für das künftige Internet (MAKI)* der *Deutschen Forschungsgemeinschaft (DFG)* (https://www.maki.tu-darmstadt.de).

Weckesser, Markus Tobias
Automatisierte Analyse integrierter Software-Produktlinien-Spezifikationen
Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
URN: urn:nbn:de:tuda-tuprints-88750
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/8875
Tag der mündlichen Prüfung: 28.06.2019
Veröffentlicht unter CC BY-SA 4.0 International
https://creativecommons.org/licenses/

ABSTRACT

Emerging trends for digital transformations (e.g., Smart Factories, Internet of Things, Smart Grids) lead to new usage scenarios that require software systems being able to reconfigure themselves continuously to meet changing environmental conditions. Integrated software product line specifications allow for precisely describing consistency properties of such systems in a uniform representation. To this end, the specification language Clafer provides means for both characterizing structural run-time variability as well as specifying re-configurable elements of the system architecture and complex dependencies. For this purpose, Clafer combines UML-like class- and meta-modeling with feature-oriented variability modeling and first-order logic constraints. This considerable expressiveness leads to complex specifications in practice (e.g. due to hidden dependencies between configuration options and components of the system architecture). As a result, integrated software product line specifications tend to be prone to specification errors such as inconsistencies or anomalies (e.g., dead configuration options that can never be selected). However, inconsistencies and anomalies should be identified and rectified as early as possible in the development process to avoid critical system states and involved follow-up costs in case of system failures. For this reason, static analysis techniques for automated validation of integrated product line specifications are crucially needed.

Recent analysis techniques for checking model consistency of integrated product line specifications impose an a priori finite search space with either manually or heuristically adjusted bounds. However, these techniques are inherently incomplete thus yielding results for which it is unknown whether initially chosen bounds are sufficient and, therefore, have limited practical applicability. Furthermore, no techniques for automated anomaly detection in Clafer specifications that have been proposed so far go beyond inconsistency analysis and allow for identifying more sophisticated anomaly types (as e.g., dead multiplicity intervals). Moreover, existing approaches for consistency checking provide only limited support for analyzing specifications with complex integer attributes and almost no support for real-valued attributes.

In this thesis, we present an approach for automated analysis of integrated product line specifications that are specified in Clafer. To this end, we present a uniform specification of structural consistency properties of self-adaptive systems and lift existing notions of more sophisticated anomaly types to Clafer specifications. To tackle the completeness problem, we present a restricted, yet sufficiently expressive sublanguage of Clafer for which complete and correct consistency checks are feasible. Additionally, we introduce a novel semantic representation as a mathematical optimization problem enabling the efficient analysis of practically relevant specifications and allowing to apply well-established off-the-shelf solvers. The methods and techniques presented in this thesis are illustrated by means of a running example from the communication network domain. Furthermore, we provide a prototypical implementation of the presented approach. Our experimental evaluation shows remarkable improvements of run-time efficiency as well as effectiveness of anomaly detection as compared to existing techniques.

KURZFASSUNG

Der Trend zur Digitalisierung führt zu neuen Anwendungsszenarien (z. B. Industrie 4.0, Internet der Dinge, intelligente Stromnetze), die laufzeitadaptive Software-Systeme erfordern, die sich durch kontinuierliche Rekonfiguration an verändernde Umgebungsbedingungen anpassen. Integrierte Software-Produktlinien-Spezifikationen ermöglichen die präzise Beschreibung von Konsistenzeigenschaften derartiger Systeme in einer einheitlichen Repräsentation. So bietet die Spezifikationssprache Clafer sowohl Sprachmittel zur Charakterisierung der Laufzeitvariabilität eines Systems als auch für die rekonfigurierbaren Bestandteile der Systemarchitektur sowie komplexer Abhängigkeiten. In Clafer-Spezifikationen werden hierzu Sprachkonstrukte aus UML-Klassendiagrammen und Meta-Modellierungssprachen zusammen mit Feature-orientierten Modellierungstechniken und Constraints in Prädikatenlogik erster Stufe kombiniert. Durch die beträchtliche Ausdrucksstärke neigen derartige integrierte Produktlinien-Spezifikationen in der Praxis dazu, sehr komplex zu werden (z. B. aufgrund versteckter Abhängigkeiten zwischen Konfigurationsoptionen und Komponenten). Sie sind daher äußerst anfällig für Spezifikationsfehler in Form von Inkonsistenzen oder Entwurfsschwächen in Form von Anomalien. Inkonsistenzen und Anomalien müssen jedoch möglichst früh im Entwurfsprozess erkannt und behoben werden, um drastische Folgekosten zur Laufzeit eines Systems zu vermeiden. Aus diesem Grund sind statische Analysetechniken zur automatisierten Analyse integrierter Software-Produktlinien-Spezifikationen unabdingbar.

Existierende Ansätze zur Konsistenzprüfung erfordern, dass der Suchraum für die Instanzsuche vorab entweder manuell oder durch heuristisch identifizierte Schranken eingeschränkt wird. Da, falls keine Instanz gefunden werden kann, nicht bekannt ist, ob dies durch einen zu klein gewählten Suchraum oder eine tatsächliche Inkonsistenz verursacht wurde, sind existierende Analyseverfahren inhärent unvollständig und praktisch nur eingeschränkt nutzbar. Darüber hinaus wurden bisher noch keine Analysen zur Identifikation von Anomalien vorgeschlagen, wie sie beispielsweise in Variabilitätsmodellen auftreten können. Weiterhin erlauben existierende Verfahren zwar die Handhabung von ganzzahligen Attributen, ermöglichen jedoch keine effiziente Analyse von Spezifikationen die zusätzlich reellwertige Attribute aufweisen.

In dieser Arbeit präsentieren wir einen Ansatz zur automatisierten Analyse integrierter Software-Produktlinien-Spezifikationen, die in der Sprache Clafer spezifiziert sind. Hierfür präsentieren wir eine ganzheitliche Spezifikation der strukturellen Konsistenzeigenschaften laufzeitadaptiver Software-Systeme und schlagen neuartige Anomalietypen vor, die in Clafer-Spezifikationen auftreten können. Wir charakterisieren eine Kernsprache, die eine vollständige und korrekte Analyse von Clafer-Spezifikationen ermöglicht. Wir führen zusätzlich eine neuartige semantische Repräsentation als mathematisches Optimierungsproblem ein, die über die Kernsprache hinaus eine effiziente Analyse praxisrelevanter Clafer-Spezifikationen ermöglicht und die Anwendung etablierter Standard-Lösungsverfahren erlaubt. Die Methoden und Techniken dieser Arbeit werden anhand eines durchgängigen Beispiels eines selbst-adaptiven Kommunikationssystems illustriert und prototypisch implementiert. Die experimentelle Evaluation zeigt die Effektivität unseres Analyseverfahrens sowie erhebliche Verbesserungen der Laufzeiteffizienz im Vergleich zu etablierten Verfahren.

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

BDD	Binary Decision Diagram
DSPL	Dynamische Software-Produktlinie
CSP	Constraint Satisfaction Problem
CVL	Common Variability Language
ER	Entity-Relationship
FBBT	Feasibility-Based Bound Tightening
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
OCL	Object Constraint Language
SAT	Propositional Satisfiability Problem
SPL	Software-Produktlinie
UML	Unified Modeling Language
VAN	Vehicle Area Network

1

EINLEITUNG

Neuartige Anwendungsszenarien wie beispielsweise die Digitalisierung von Produktionsabläufen [145], Internet der Dinge [152], intelligente Stromnetze [56], dynamische Ressourcen-Allokation in Datenzentren [27, 158], die Steuerung cyber-physikalischer Systeme [30, 59] oder drahtlose Sensornetze [23] steigern die Komplexität moderner Kommunikationssysteme. Mit zunehmender Komplexität, insbesondere verursacht durch einen erhöhten Grad der Vernetzung, steigen gleichzeitig die Anforderungen an die Fähigkeit dieser Systeme, sich zur Laufzeit kontinuierlich an veränderliche Umweltbedingungen anzupassen. Ziel dieser Anpassung ist es, gleichbleibende Qualitätseigenschaften (wie z. B. Latenz oder Energiebedarf) und konsistente Systemzustände zu gewährleisten [85]. Um die Anpassungsfähigkeit an Anforderungen sicherzustellen, die sich zur Laufzeit verändern können, muss die Architektur solcher Systeme Variabilität zur Entwurfszeit vorsehen, sodass beispielsweise eine Anpassung von Parametern oder der Austausch einzelner Komponenten oder ganzer Komponentenstrukturen zur Laufzeit ermöglicht wird [22].

Selbst-adaptive Systeme charakterisieren eine Klasse von Systemen, die sich zur Laufzeit anpassen können. Unter dem Begriff der Selbst-*-Eigenschaften sind Zielsetzungen kategorisiert, die mit dem Einsatz von selbst-adaptiven Software-Systemen verfolgt werden [72, 83]. Dazu gehören die übergeordneten Eigenschaften Selbst-Wahrnehmung [70] (engl. self-awareness) und Kontext-Wahrnehmung (engl. context awareness) sowie die untergeordneten Eigenschaften Selbst-Konfiguration (engl. self-configuration), Selbst-Heilung (engl. self-healing), Selbst-Optimierung (engl. self-optimization) und Selbst-Schutz (engl. self-protection). Eine wesentliche Charakteristik selbst-adaptiver Systeme ist die Unterscheidung (i) der Adaptionslogik, die diejenigen Aktivitäten zusammenfasst, welche die Kontrolldaten einer Implementierung beeinflussen [28], sowie (ii) verwalteter Ressourcen, die Gegenstand von Adaptionsentscheidungen sind.

Bei Ansätzen mit externer Adaptionslogik werden Kontrolldaten zur Steuerung von Adaptionen getrennt von der zu adaptierenden verwalteten Ressource betrachtet [129]. Zur Strukturierung der externen Adaptionslogik hat sich als grundlegende Referenzarchitektur die MAPE-Schleife etabliert [83]. Hierbei werden Komponenten (i) zum Erfassen von Umgebungsdaten (engl. monitoring), (ii) zur Analyse der Daten und zur Inferenz des Kontextzustandes (engl. analysis), (iii) zur Planung konsistenter und zielsetzungsadäqua-

ter Rekonfigurationsentscheidungen (engl. planning) und (iv) zur Ausführung dieser Entscheidungen (engl. execution) unterschieden, die sukzessive über den Lebenszyklus eines Adaptionsprozesses hinweg durchlaufen werden.

Bei modellbasierten Planungskomponenten werden die Architektur, das Verhalten sowie der Kontext der verwalteten Ressourcen in Form von Modellen erfasst [72, 89]. Der aktuelle Systemzustand wird als Instanz der Modelle repräsentiert. Modelle beschreiben dabei wesentliche Konsistenzeigenschaften des Systems (z. B. Einschränkungen der Adaptionsmöglichkeiten für bestimmte Systemkontexte). Verletzungen der Konsistenzeigenschaften können festgestellt und durch Rekonfigurationsentscheidungen behoben werden [107]. Durch Verwendung eines modellbasierten Ansatzes können Inkonsistenzen des Systems zur Laufzeit vermieden werden. Dies setzt allerdings voraus, dass die Spezifikationen der zur Laufzeit verwendeten Modelle selbst keine Inkonsistenzen aufweisen (z. B. dass für alle Umgebungssituationen adäquate Rekonfigurationsentscheidungen getroffen werden können).

Software-Produktlinien (SPLs) stellen Technologien und Methoden zur Spezifikation und Analyse variabler Systemarchitekturen bereit [112]. In einem strukturierten Entwicklungsprozess werden Gemeinsamkeiten und Unterschiede zwischen Software-Varianten durch Konfigurationsoptionen charakterisiert und Abhängigkeiten in Form von Variabilitätsmodellen spezifiziert [40]. In einem Feature-orientierten Entwicklungsprozess werden im Rahmen des Domain-Engineering-Teilprozesses Konfigurationsoptionen als Features erfasst. Ein Feature ist definiert als nutzersichtbare Eigenschaft, die geeignet ist, Gemeinsamkeiten und Unterschiede von Software-Varianten einer SPL zu charakterisieren [7]. Ein Menge selektierter Features wird als Konfiguration bezeichnet und repräsentiert eine Software-Variante. Entsprechend charakterisieren Feature-Modelle Abhängigkeiten zwischen Features und repräsentieren auf diese Weise den Raum zulässiger Konfigurationen. Zur Spezifikation von Feature-Modellen stellen Feature-Diagramme eine baumartige visuelle Syntax bereit [78].

Dynamische Software-Produktlinien (DSPLs) erweitern den Entwicklungsprozess zusätzlich um Möglichkeiten der Spezifikation und Analyse von Laufzeitvariabilität und stellen eine etablierte Methodik zur Konzeption selbst-adaptiver Systeme im Allgemeinen [23, 64] und zum systematischen Entwurf modellbasierter Planungskomponenten im Speziellen dar [111, 146]. Hinsichtlich des Verwendungszwecks von Spezifikationen einer DSPL wird zwischen den Perspektiven Problemraum und Lösungsraum unterschieden [43]. Der Problemraum umfasst Variabilitätsmodelle, die dynamische (kontextabhängige) Rekonfigurationsoptionen der Adaptionslogik beschreiben (z. B. als Kontext-Feature-Modellen [66, 104, 130]). Der Lösungsraum umfasst Implementierungsartefakte (z. B. die Architekturbeschreibung als von Klassendiagramm der Unified Modeling Language (UML) [128]), die innerhalb eines Rekonfigurationsprozesses entsprechend der gewählten Konfigurationsoptionen und unter Berücksichtigung der Lösungsraumspezifikation instanziiert werden.

Zur Spezifikation von Abhängigkeiten zwischen Konfigurationsoptionen des Problemraums und Implementierungsartefakten des Lösungsraums können Variabilitätsannotationen [42], gesonderte Zuordnungmodelle [47, 67] oder integrierte Produktlinien-Spezifikationen verwendet werden. Integrierte Produktlinien-Spezifikationen beschreiben Problem- und Lösungsraumartefakte in der gleichen Spezifikationssprache und erlauben so die Beschreibung komplexer Abhängigkeiten in einer einheitlichen Repräsentation. Die Sprache Clafer [10] stellt einen wichtigen Vertreter integrierter Software-Produktlinien-Spezifikationen dar. Mithilfe von Clafer können Variabilitätsannotationen aus Problemraumspezifikationen mit Sprachkonstrukten struktureller Spezifikationssprachen (z. B. Klassen, Vererbung und Referenzen) zusammen mit Constraints, welche die Ausdrucksstärke von Prädikatenlogik erster Stufe haben, kombiniert werden. Clafer ermöglicht im Gegensatz zu annotativen Ansätzen oder gesondert spezifizierter Zuordnungsmodelle eine geschlossene Validierung und Analyse, ohne Spezifikationen zuvor zusammenführen zu müssen.

Eine wesentliche Motivation für den Einsatz modellbasierter Ansätze zur Konzeption selbst-adaptiver Systeme ist die Möglichkeit, bereits zur Entwurfszeit inkonsistente Systemzustände zu identifizieren, die im schlimmsten Fall zur Laufzeit zum Ausfall des Systems führen [38]. Da die Spezifikationen in der Praxis dazu neigen sehr komplex zu werden (z. B. aufgrund versteckter Abhängigkeiten zwischen Problem- und Lösungsraumartefakten), sind sie äußerst anfällig für Spezifikationsfehler in Form von Inkonsistenzen oder Entwurfsschwächen in Form von Anomalien. Von entscheidender Bedeutung ist zudem, dass Inkonsistenzen und Anomalien in den Spezifikationen so früh wie möglich in der Entwurfsphase eines Systems erkannt und behoben werden, um drastische Folgekosten zur Laufzeit zu vermeiden, die im schlimmsten Fall durch den Ausfall ganzer Infrastrukturen entstehen [138]. Aus diesem Grund sind Ansätze und Werkzeugunterstützung zur automatisierten Analyse integrierter Software-Produktlinien-Spezifikationen unabdingbar.

1.1 WISSENSCHAFTLICHE HERAUSFORDERUNGEN

In diesem Abschnitt charakterisieren wir wissenschaftliche Herausforderungen, die sich bei der Validierung und Analyse integrierter Produktlinien-Spezifikationen ergeben und die im Rahmen der vorliegenden Arbeit behandelt werden.

Herausforderung 1: *Spezifikation von Konsistenzeigenschaften selbst-adaptiver Systeme*

Die Validierung von Konsistenzeigenschaften und die Erkennung potentieller Anomalien in selbst-adaptiven Systemen erfordern eine geeignete Repräsentation der Konfigurationsoptionen im Problemraum und der zu adaptierenden verwalteten Ressourcen im Lösungsraum. Inkonsistenzen entstehen häufig erst durch komplexe Abhängigkeiten zwischen Problem- und Lösungsraumartefakten. Es genügt daher nicht Spezifikatio-

nen des Problem- und Lösungsraums getrennt zu betrachten. Implizit vorhandene Abhängigkeiten zwischen Problem- und Lösungsraumelementen können zur Laufzeit zu fehlerhaften Systemzuständen führen, die potentiell drastische Auswirkungen auf das Systemverhalten haben [36, 118]. Die explizite Spezifikation der Abhängigkeiten zwischen Elementen des Problem- und Lösungsraums ist somit wesentliche Voraussetzung für die Identifikation von Inkonsistenzen und Anomalien. Dabei stellen sich jedoch folgende Herausforderungen:

(i) *Multiinstanziierung:* Problem- und Lösungsraumelemente können potentiell mehrfach instanziiert sein. So kann beispielsweise eine Komponente der Lösungsraumarchitektur in verschiedenen Instanzen auftreten, wobei jede Instanz der Komponente individuell konfiguriert ist. Zur Spezifikation derartiger Abhängigkeiten ist es erforderlich, Instanzen von Konfigurationsoptionen zu charakterisieren (z. B. durch kardinalitätsbasierte Feature-Modellen [45]) und diese durch lokale Einschränkungen an Komponenteninstanzen zu binden. Darüber hinaus sind globale Einschränkungen erforderlich, um Abhängigkeiten zwischen Typen von Konfigurationsoptionen zu spezifizieren [41]. Zur Entwurfszeit kann häufig keine Aussage gemacht werden, über wie viele Instanzen eine Konfigurationsoption oder eine Komponente verfügt (z. B. da in einem Kommunikationssystem Komponenten, die Kommunikationsknoten repräsentieren, potentiell beliebig oft auftreten können). Es sind daher Sprachmittel erforderlich, um eine potentiell beliebig große Instanzmenge zu spezifizieren.

(ii) *Orthogonale (rekursive) Vererbungs- und Kompositionsbeziehungen:* Eine Komponente kann selbst aus anderen Unterkomponenten bestehen, die austauschbare Module mit gemeinsamen Schnittstellen repräsentieren. Konfigurationsoptionen, die an (abstrakte) Schnittstellen gebunden sind, müssen an konkrete Instanzen der Komponenten propagiert werden. Es entstehen somit komplexe Abhängigkeiten, ausgelöst durch die Vermischung von Vererbungs- und Kompositionsbeziehungen.

(iii) *Komplexe Attribut-Einschränkungen:* Zur Spezifikation der Abhängigkeiten von Konfigurationsoptionen zur jeweiligen Systemumgebung wurden Kontext-Feature-Modelle vorgeschlagen [66, 104, 130]. Die Variabilität der Systemumgebung wird hierbei in Form binärer Konfigurationsoptionen des Kontexts erfasst und durch Constraints in Beziehung zu (binären) Konfigurationsoptionen des Systems gesetzt. Auf diese Weise lassen sich Konsistenzeigenschaften zwischen Systemumgebung (repräsentiert als Belegung von Konfigurationsoptionen des Kontexts) und System (repräsentiert als Belegung von Konfigurationsoptionen des Systems) spezifizieren.

Bei der beschriebenen Modellierungstechnik liegt der Fokus auf dem Problemraum: Es werden die Konfigurationsoptionen des Kontexts entsprechend der jeweiligen Systemumgebung im Rahmen der Analyse-Phase der MAPE-Schleife vorbelegt, ohne Aussagen zu treffen wie (ganzzahlige oder reellwertige) Sensordaten der

Monitoring-Phase Konfigurationsoptionen des Kontexts zugeordnet sind. Bei einer ganzheitlichen Betrachtung von Problem- und Lösungsraum ist es hingegen zusätzlich erforderlich, Abhängigkeiten zwischen Sensordaten (bereitgestellt durch Lösungsraumelemente) und Konfigurationsoptionen des Kontexts zu spezifizieren.

Herausforderung 2: *Identifikation von Inkonsistenzen und Anomalien*

Eine Software-Anomalie wird im IEEE-Standard 1044-2009 definiert als Irregularität, Inkonsistenz oder sonstige Abweichung von Erwartungen hinsichtlich Verhalten, Form oder Funktion [74]. Ein Variabilitätsmodell ist *inkonsistent*, wenn es keine valide Konfiguration besitzt, die alle spezifizierten Constraints erfüllt. Beispielsweise können in Feature-Modellen widersprüchliche Constraints zwischen Features dazu führen, dass keine gültige Konfiguration existiert. Ein Variabilitätsmodell besitzt eine *Anomalie*, wenn es Konfigurationsoptionen auf syntaktischer Ebene vorsieht, die auf semantischer Ebene aber durch die angegebenen Constraints ausgeschlossen wird. Beispielsweise liegt für ein Feature eines Feature-Modells eine Anomalie vom Typ *totes Features* (engl. dead feature) vor, falls keine zulässige Konfiguration existiert, in der das Feature auftritt, obwohl es syntaktisch in der Spezifikation angegeben ist. Für Feature-Modelle wurden einige Analysetechniken zur Identifikation von Anomalien präsentiert [20, 57]. Die meisten in der Literatur vorgeschlagenen Validierungstechniken unterstützen die Analyse von Feature-Modellen, die Konfigurationen mit ausschließlich binären Features repräsentieren (ein Feature kann in einer Konfiguration entweder an- oder abgewählt sein).

Zusätzlich wurden für Feature-Modelle zwei Erweiterungen der Konfigurationssemantik vorschlagen, welche die Ausdrucksstärke der Spezifikationen zwar erhöhen, jedoch die Analyse herausfordernder machen: (i) Nicht-binäre Feature-Typen [43, 78] und (ii) Multiplizitätsintervalle [125]. Nicht-binäre Feature-Typen erlauben die Repräsentation ganzzahliger oder reellwertiger Attribute in einer Konfiguration. Durch die Erweiterung der Spezifikation um Multiplizitätsintervalle in kardinalitätsbasierten Feature-Modellen können Konfigurationen repräsentiert werden, die mehrere Instanzen eines Feature-Typs enthalten. Ein Multiplizitätsintervall gibt an, wie viele Instanzen eines Feature-Typs mindestens oder höchstens in einer Konfiguration auftreten dürfen. Falls ein Multiplizitätsintervall unbeschränkt ist, können potentiell beliebig viele Feature-Instanzen in einer Konfiguration selektiert werden. Ein kardinalitätsbasiertes Feature-Modell, in dem unbeschränkte Multiplizitätsintervalle auftreten, kann somit eine potentiell unbeschränkte Menge von Konfigurationen repräsentieren. Ein Feature, das mehrere Instanzen in einer Konfiguration aufweist, wird als multiinstanziiert bezeichnet. Insbesondere die Erweiterung der Konfigurationssemantik um Multiinstanziierung in kardinalitätsbasierten Feature-Modellen stellt eine vielversprechende Erweiterung zur Spezifikation verteilter selbst-adaptiver Systeme dar. Multiplizitätsintervalle aus kardinalitätsbasierten Feature-Modellen wurden unter anderem in die Spezifikationssprachen Clafer [9] und Common Variability Language (CVL) [67] übernommen. Beide genannten Erweiterungen erhöhen

jedoch die Komplexität der Konfigurationssemantik gegenüber Feature-Modellen mit binären Konfigurationsoptionen und erfordern daher zusätzliche Analysetechniken für die Aufdeckung von Inkonsistenzen und Anomalien.

Bei kardinalitätsbasierten Feature-Modellen treten neue Anomalietypen auf. Eine Anomalie vom Typ *totes Kardinalitätsintervall* (engl. dead cardinality interval) liegt bei einem Multiplizitätsintervall beispielsweise vor, falls für ein betrachtetes Teilintervall keine zulässige Konfiguration gefunden werden kann. Für Feature-Modelle mit ganzzahligen numerischen Features mit beschränkten Wertebereichen wurden Analysetechniken zur Identifikation von Inkonsistenzen und Anomalien vorgeschlagen [19, 29, 80]. Jedoch existieren erst Ansätze für die Analyse kardinalitätsbasierter Feature-Modelle mit beschränkten Multiplizitätsintervallen [45, 105, 117, 119] und kardinalitätsbasierter Feature-Modelle mit ganzzahligen Attributen mit beschränkten Multiplizitätsintervallen und beschränkten Wertebereichen [41]. Für erweiterte kardinalitätsbasierte Feature-Modelle mit (i) Multiplizitätsintervallen, die potentiell unbeschränkt sind, (ii) mit ganzzahligen numerischen Features mit beschränkten Wertebereichen oder (iii) mit reellwertigen numerischen Features wurden hingegen bisher keine Analysetechniken präsentiert. Zudem können durch unbeschränkte Multiplizitätsintervalle und unbeschränkte Wertebereiche numerischer Features neuartige Typen von Anomalien auftreten: Beispielsweise liegt eine Anomalie vom Typ *fälschlicherweise unbeschränktes Kardinalitätsintervall* vor, falls ein Multiplizitätsintervall, das als unbeschränkt spezifiziert wurde, aufgrund von Seiteneffekten tatsächlich beschränkt ist.

Für integrierte Software-Produktlinien-Spezifikationen, die in der Sprache Clafer formuliert sind, wurden über Konsistenzprüfungen von Spezifikationen hinaus keine vergleichbaren Analysen präsentiert [6]. Aufgrund der beträchtlichen Ausdrucksstärke von Clafer und wegen der komplexen Semantik sind Analysetechniken zur Identifikation von Anomalien umso wichtiger, jedoch herausfordernd.

Herausforderung 3: *Effiziente Analyse von Constraints über ganzzahlige und reellwertige Attribute*

In Spezifikationen selbst-adaptiver Systemen spielen ganzzahlige und reellwertige Attribute eine wichtige Rolle: Sie werden unter anderem verwendet, um Kontextinformationen im Problemraum (z. B. Latenz einer Kommunikationsverbindung in Millisekunden) oder sonstige Parameter von Komponenten im Lösungsraum zu spezifizieren. Bestehende Ansätze zur Analyse von Spezifikationen mit ganzzahligen Attributen codieren das zugrundeliegende Analyseproblem meist als Binary Decision Diagram (BDD) [159], als Erfüllbarkeitsproblem (engl. Propositional Satisfiability Problem (SAT)) [16, 93, 102] oder als Constraint-Erfüllbarkeitsproblem (engl. Constraint Satisfaction Problem (CSP)) [19]. Attribute werden bei den genannten Ansätzen aussagenlogischen Ausdrücken zugeordnet. Dies ist jedoch nur möglich bei Attributen mit endlichen Wertebereichen und erfordert eine Diskretisierung reellwertiger Attribute. Bei Attributen mit sehr großen

Wertebereichen und bei komplexen Constraints zwischen Attributen ist eine effiziente Analyse bei keinem der genannten Vorgehen möglich. Existierende Verfahren zur Analyse integrierter Produktlinien-Spezifikationen, die in der Sprache Clafer vorliegen, unterstützen keine reellwertigen Attribute [6]. Aufgrund ganzzahliger und reellwertiger Attribute mit potentiell unbeschränkten Wertebereichen, die in integrierten Produktlinien-Spezifikationen selbst-adaptiver Systeme vorherrschend sind, ist die effiziente Analyse derartiger Spezifikationen herausfordernd.

Herausforderung 4: *Existierende Ansätze zur Konsistenzprüfung nicht vollständig*

Existierende Verfahren zur Konsistenzprüfung integrierter Produktlinien-Spezifikationen in der Sprache Clafer codieren das zugrundeliegende Analyseproblem entweder in der Spezifikationssprache Alloy [76] oder verwenden eine Formulierung als Constraint-Programming-Problem [6]. Für beide Ansätze ist es erforderlich, dass für die Anzahl von Modellelementen eines Typs obere Schranken manuell oder heuristisch festgelegt werden, um den Suchraum zu beschränken, in dem nach gültigen Spezifikationsinstanzen gesucht wird. Dies führt zu langwierigen und fehleranfälligen Iterationsschleifen, in denen der Suchraum (zumeist manuell) angepasst werden muss. Insbesondere ist dieses Vorgehen nicht praktikabel, falls Clafer als Zwischenrepräsentation verwendet wird, um beispielsweise zur Laufzeit gültige Architekturinstanzen zu ermitteln, oder wenn sie als Zielrepräsentation einer domänenspezifischen Sprache fungiert [84].

Falls innerhalb des manuell oder durch eine Heuristik festgelegten Suchraums eine Spezifikationsinstanz durch die Analysetechniken gefunden wird, kann geschlossen werden, dass die Spezifikation konsistent ist. Falls hingegen keine gültige Spezifikationsinstanz gefunden wird, kann dies darauf zurückgeführt werden, dass entweder (i) die Schranken zur Begrenzung des Suchraums zu klein gewählt wurden oder (ii) tatsächlich keine gültige Spezifikationsinstanz existiert. In diesem Sinne sind existierende Analysetechniken nicht vollständig.

1.2 ZIELSETZUNG UND BEITRAG

Die übergeordnete Zielsetzung dieser Arbeit ist die Entwicklung eines Ansatzes zur automatisierten Analyse integrierter Produktlinien-Spezifikationen, die in der Sprache Clafer spezifiziert sind. Im Folgenden beschreiben wir, wie sich dieses Ziel in einzelne Forschungsziele gliedert. Zu jedem Forschungsziel präsentieren wir die damit verbundenen erreichten wissenschaftlichen Beiträge (WB).

Forschungsziel 1: Identifikation von Anomalien in Clafer-Spezifikationen

In dieser Arbeit stellen wir anhand des Beispiels eines selbst-adaptiven Kommunikationssystems die Spezifikation von Konsistenzeigenschaften von Clafer vor. Dabei demonstrieren wir die Spezifikation multiinstanziierbarer Komponenten und Konfigurationsoptionen, orthogonaler (rekursiver) Vererbungs- und Kompositionsbeziehungen sowie komplexer Attribut-Constraints. Ausgehend von existierenden Anomalietypen, die in Spezifikationen des Problemraums auftreten, definieren und charakterisieren wir neuartige Anomalietypen, die in Clafer-Spezifikationen auftreten können und potentiell zu schwerwiegenden Problemen zur Laufzeit des Systems führen. Einige der neuartigen Anomalietypen beziehen sich hierbei auf Inkonsistenzen, die durch Modellelemente resultieren, die als beliebig oft instanziierbar spezifiziert sind, jedoch tatsächlich nur in begrenzter Anzahl in der Spezifikation auftreten. Bezüglich des Forschungsziels 1 liefert diese Arbeit folgende wissenschaftliche Beiträge:

WB 1: Ganzheitliche Spezifikation der strukturellen Eigenschaften und Konsistenzeigenschaften eines selbst-adaptiven Systems in der Sprache Clafer.

WB 2: Identifikation und Charakterisierung neuartiger Anomalietypen, die in Clafer-Spezifikationen auftreten können.

Forschungsziel 2: Vollständige und korrekte Analyse von Clafer-Spezifikationen

Eine Clafer-Spezifikation ist inkonsistent, falls keine Spezifikationsinstanz existiert, die alle Constraints erfüllt. Weiterhin weist eine Clafer-Spezifikation eine Anomalie auf, falls Modellelemente syntaktisch spezifiziert sind, jedoch aufgrund von Constraints auf semantischer Ebene ausgeschlossen werden. Zur Analyse von Clafer-Spezifikationen auf Inkonsistenzen und Anomalien definieren und charakterisieren wir in dieser Arbeit eine Kernsprache, die Teilmenge der Sprache Clafer ist. Für Spezifikationen, die in der Kernsprache vorliegen, ermöglicht unser Analyseverfahren, dass jede Inkonsistenz und Anomalie entdeckt wird. Die in dieser Arbeit vorgeschlagene Kernsprache ermöglicht somit *vollständige* Analysen von Clafer-Spezifikationen. Im Vergleich zu existierenden Verfahren müssen für Modellelemente keine oberen Schranken angegeben werden (soweit die Spezifikation in der Kernsprache vorliegt), um den Raum für die Suche nach validen Instanzen zu begrenzen. Falls keine Instanz für eine Spezifikation durch unser Analyseverfahren gefunden wird, kann somit sicher geschlossen werden, dass die ursprüngliche Spezifikation tatsächlich inkonsistent ist.

Darüber hinaus führen wir eine erweiterte Kernsprache ein, die im Sprachumfang gegenüber der Kernsprache ausdrucksstärker ist. Jede durch unser Analyseverfahren entdeckte Inkonsistenz oder Anomalie in einer Spezifikation, die in der erweiterten Kernsprache vorliegt, ist auch tatsächlich eine solche. Die in dieser Arbeit vorgeschlagene

erweiterte Kernsprache ermöglicht somit die *korrekte* Analyse von Clafer-Spezifikationen. Bezüglich des Forschungsziels 2 liefert diese Arbeit folgende wissenschaftliche Beiträge:

WB 3: Charakterisierung einer Kernsprache von Clafer, die eine vollständige und korrekte Analyse von Clafer-Spezifikationen ermöglicht.

WB 4: Charakterisierung einer *erweiterten* Kernsprache von Clafer, die eine korrekte Analyse von Clafer-Spezifikationen ermöglicht.

Forschungsziel 3: Effiziente und automatisierte Analyse von Clafer-Spezifikationen

Zur effizienten Analyse von Clafer-Spezifikationen präsentieren wir eine neuartige Multimengen-Repräsentation. Diese Repräsentation codiert die Semantik der Sprache Clafer und erlaubt Abschätzungen über die Anzahl von Instanzen von Sprachelementen in Spezifikationsinstanzen. Die Multimengen-Repräsentation ermöglicht es, Spezifikationen in Bezug auf die zuvor charakterisierten Anomalietypen zu analysieren. Die Analyseziele und die Multimengen-Repräsentation einer Clafer-Spezifikation werden hierzu als mathematisches Optimierungsproblem formuliert. Dies ermöglicht eine effiziente Handhabung von ganzzahligen und reellwertigen Attributen und erlaubt darüber hinaus die Anwendung von industrieerprobten Standard-Lösungsverfahren. Bezüglich des Forschungsziels 3 liefert diese Arbeit folgende wissenschaftliche Beiträge:

WB 5: Eine neuartige semantische Repräsentation, die über die Kernsprache hinaus eine effiziente Analyse von Clafer-Spezifikationen ermöglicht.

WB 6: Eine Repräsentation der Analyseziele als mathematische Optimierungsprobleme, welche die Anwendung von Standard-Lösungsverfahren erlaubt.

Der Fokus dieser Arbeit liegt auf der Analyse von Architekturen sowie struktureller Eigenschaften selbst-adaptiver Systeme und der Analyse von Zuordnungen von Komponenten bzw. Teilarchitekturen zu Konfigurationsoptionen in Form integrierter Produktlinien-Spezifikationen in der Sprache Clafer. Zeitliche und verhaltensorientierte Aspekte selbst-adaptiver Systeme werden hierbei nicht explizit einbezogen. Eine kürzlich vorgeschlagene Erweiterung der Sprache Clafer zur Spezifikation verhaltensorientierter Aspekte [77] stellt daher eine vielversprechende Richtung für weitere Forschungsaktivitäten dar.

1.3 AUFBAU DER ARBEIT

Im Folgenden beschreiben wir den Aufbau der Arbeit und ordnen den Kapiteln die wissenschaftlichen Beiträge zu.

- In Kapitel 2 führen wir grundlegende Konzepte von Software-Produktlinien und Dynamischen Software-Produktlinien ein, die zum Verständnis dieser Arbeit erfor-

derlich sind. Dies erfolgt anhand des motivierenden Beispiels eines selbst-adaptiven Kommunikationssystems.

- In Kapitel 3 stellen wir die Spezifikation des selbst-adaptiven Kommunikationssystems als integrierte Software-Produktlinie in der Sprache Clafer vor und erläutern anhand der Spezifikation die unterschiedlichen Sprachelemente der Sprache Clafer. Darüber hinaus charakterisieren wir anhand der Spezifikation neuartige Anomalietypen. Dieses Kapitel beschreibt die wissenschaftlichen Beiträge **WB 1** und **WB 2**.

- In Kapitel 4 führen wir die Syntax und Wohlgeformtheitseigenschaften von Clafer-Spezifikationen ein. Zusätzlich formalisieren wir Korrektheits- und Vollständigkeitseigenschaften, die bei Analyse der Clafer-Spezifikationen gefordert werden. Dieses Kapitel liefert Anteile zu den wissenschaftlichen Beiträgen **WB 3** und **WB 4**.

- In Kapitel 5 präsentieren wir eine neuartige Multimengen-Repräsentation integrierter Produktlinien-Spezifikationen, die in der Sprache Clafer spezifiziert sind. Dazu beschreiben wir die Transformation der Sprachelemente von Clafer in die Multimengen-Repräsentation. Dieses Kapitel beschreibt den wissenschaftlichen Beitrag **WB 5**.

- In Kapitel 6 präsentieren wir, wie auf Basis der Multimengen-Repräsentation Clafer-Spezifikationen automatisiert analysiert werden können. Zunächst charakterisieren wir dazu die beiden betrachteten Teilsprachen von Clafer: Die Kernsprache, die eine vollständige und korrekte, und die erweiterte Kernsprache, die eine korrekte Analyse von Clafer-Spezifikationen ermöglicht. In einem zweiten Schritt präsentieren wir, wie Analyseziele als mathematische Optimierungsprobleme formuliert und mithilfe etablierter Standard-Lösungsverfahren analysiert werden können. Dieses Kapitel beschreibt den wissenschaftlichen Beitrag **WB 6** und liefert Anteile zu den wissenschaftlichen Beiträgen **WB 3** und **WB 4**.

- In Kapitel 7 führen wir eine experimentelle Evaluation des präsentierten Analyseverfahrens bezüglich Effektivität und Effizienz durch. Zusätzlich beschreiben wir die Implementierung, die eine Werkzeugunterstützung zur automatisierten Analyse von Clafer-Spezifikationen bereitstellt. Diese Kapitel liefert Anteile zu den wissenschaftlichen Beiträgen **WB 3** bis **WB 4**.

- In Kapitel 8 diskutieren wir verwandte Arbeiten zur Analyse von Problemraum- und Lösungsraumspezifikationen.

- In Kapitel 9 folgt eine Zusammenfassung dieser Arbeit und ein Ausblick auf vielversprechende zukünftige Forschungsfelder, die sich aus dieser Arbeit ergeben.

2

GRUNDLAGEN

In diesem Kapitel führen wir diejenigen grundlegenden Konzepte ein, die für ein tiefgreifendes Verständnis der in Abschnitt 1.1 behandelten Herausforderungen erforderlich sind. Zunächst präsentieren wir in Abschnitt 2.1 ein motivierendes Beispiel eines selbst-adaptiven Kommunikationssystems. Das Beispiel dient zur Illustration der in dieser Arbeit entwickelten Konzepte. In Abschnitt 2.2 legen wir die Grundlagen zu Software-Produktlinien (SPLs) dar und stellen Spezifikationen zur Beschreibung des Problemraums vor. In Abschnitt 2.3 führen wir Dynamische Software-Produktlinien (DSPLs) als wesentliche Erweiterung von Software-Produktlinien (SPLs) ein. Schließlich präsentieren wir in Abschnitt 2.4 die Spezifikation der Lösungsraumarchitektur.

2.1 MOTIVIERENDES UND ILLUSTRIERENDES BEISPIEL

Im Rahmen dieser Arbeit betrachten wir zur Illustration der vorgestellten Konzepte als durchgängiges Beispiel ein selbst-adaptives Kommunikationssystem, das im Rahmen des Sonderforschungsbereichs Multi-Mechanismen Adaption für das zukünftige Internet (MAKI) untersucht wurde [122, 124, 149]. Das selbst-adaptive Kommunikationssystem weist eine externe Adaptionslogik auf. Dies bedeutet, dass wir zwischen Kontrolldaten, welche die Adaptionsentscheidungen steuern, und verwalteten Ressourcen, die Messdaten liefern und Adaptionsentscheidungen umsetzen, unterscheiden [129]. Zunächst betrachten wir in folgendem Beispiel die Bestandteile und Kommunikationsmechanismen, welche die verwaltete Ressourcen des selbst-adaptiven Kommunikationssystems darstellen und somit durch Adaptionsentscheidungen beeinflussbar sind.

Beispiel 2.1 (Selbst-adaptives Kommunikationssystem: Bestandteile und Kommunikationsmechanismen)

Bestandteile des Systems: Das Kommunikationssystem besteht aus *Kommunikationsknoten*, die miteinander interagieren und zueinander *Kommunikationsverbindungen* aufbauen. Die Kommunikationsknoten und Kommunikationsverbindungen bilden ein *Kommunikationsnetz*. Abhängig von der Mobilität eines Knotens werden *Infrastrukturknoten* und mobile Knoten unterschieden. Infrastrukturknoten sind statisch

und bieten mobilen Kommunikationsknoten die Möglichkeit, Daten mit anderen (entfernten) Netzwerken auszutauschen. Ein Infrastrukturknoten repräsentiert beispielsweise Funkmasten von Telekommunikationsanbietern, statische Sensorknoten in drahtlosen Sensornetzwerken [58] oder Road Side Units in Vehicle Area Networks (VANs) [62]. Im betrachteten Beispiel sind alle Infrastrukturknoten durch ein Backend-Netzwerk verbunden. *Mobile Knoten* repräsentieren beispielsweise mobile Endgeräte [122], bewegliche Sensorknoten in drahtlosen Sensornetzwerken [58] oder Fahrzeuge in VANs [62].

Adaptierbare Kommunikationsmechanismen: Das Kommunikationssystem ermöglicht den Austausch von Daten zwischen Knoten, indem es Kommunikationsmechanismen auf verschiedenen Schichten bereitstellt. Funktional gleiche Kommunikationsmechanismen können zur Laufzeit ausgetauscht oder angepasst werden, um auf sich verändernde Systemumgebungen adäquat zu reagieren (z. B. zur Vermeidung von inkonsistenten Systemzuständen als Reaktion auf Überlastszenarien kann von einem herkömmlichen zu einem probabilistischen Broadcast-Protokoll gewechselt werden). Eine Menge funktional gleicher, jedoch qualitativ unterschiedlicher Mechanismen wird als Multimechanismus bezeichnet [4]. Wir unterscheiden im betrachteten Beispiel Multimechanismen auf physikalischer, Transport- und Anwendungsschicht:

- Auf der physikalischen Schicht stellen Knoten Kommunikationsverbindungen über Mobilfunk (LTE) oder Wi-Fi zu anderen Knoten her. Mobile Knoten kommunizieren mit Infrastrukturknoten über Mobilfunkverbindungen. Dazu benötigen sie eine aktivierte LTE-Schnittstelle. Alternativ können mobile Knoten dezentral innerhalb von Gruppen kommunizieren, indem sie Ad-hoc-Kommunikationsverbindungen via Wi-Fi aufbauen. Innerhalb einer Gruppe fungiert ein einzelner mobiler Knoten als *Relay* und vermittelt Informationen zwischen Mitgliedern der Gruppe und Infrastrukturknoten. Mobile Knoten, die als Relay fungieren, benötigen sowohl Schnittstellen zur Kommunikation mit Infrastrukturknoten (z. B. LTE) als auch Schnittstellen zur Ad-hoc-Kommunikation mit anderen mobilen Knoten (z. B. Wi-Fi). Die maximale Anzahl aktiver Verbindungen je Knoten ist beschränkt (z. B. aufgrund von Ressourcenbeschränkungen der Infrastrukturknoten oder um Interferenz bei gleichzeitigem Nachrichtenversand zu reduzieren).

- Auf der Transportschicht wird das verbindungsorientierte Transportprotokoll TCP [114] und das verbindungslose Transportprotokoll UDP [113] innerhalb eines Multimechanismus bereitgestellt. Die Wahl des Kommunikationsmechanismus hat Auswirkungen auf den Energieverbrauch der Knoten, auf die zur Verfügung stehende Bandbreite im Kommunikationsnetz und auf die Zuverlässigkeit des Nachrichtenversands. Das Transportprotokoll UDP wird

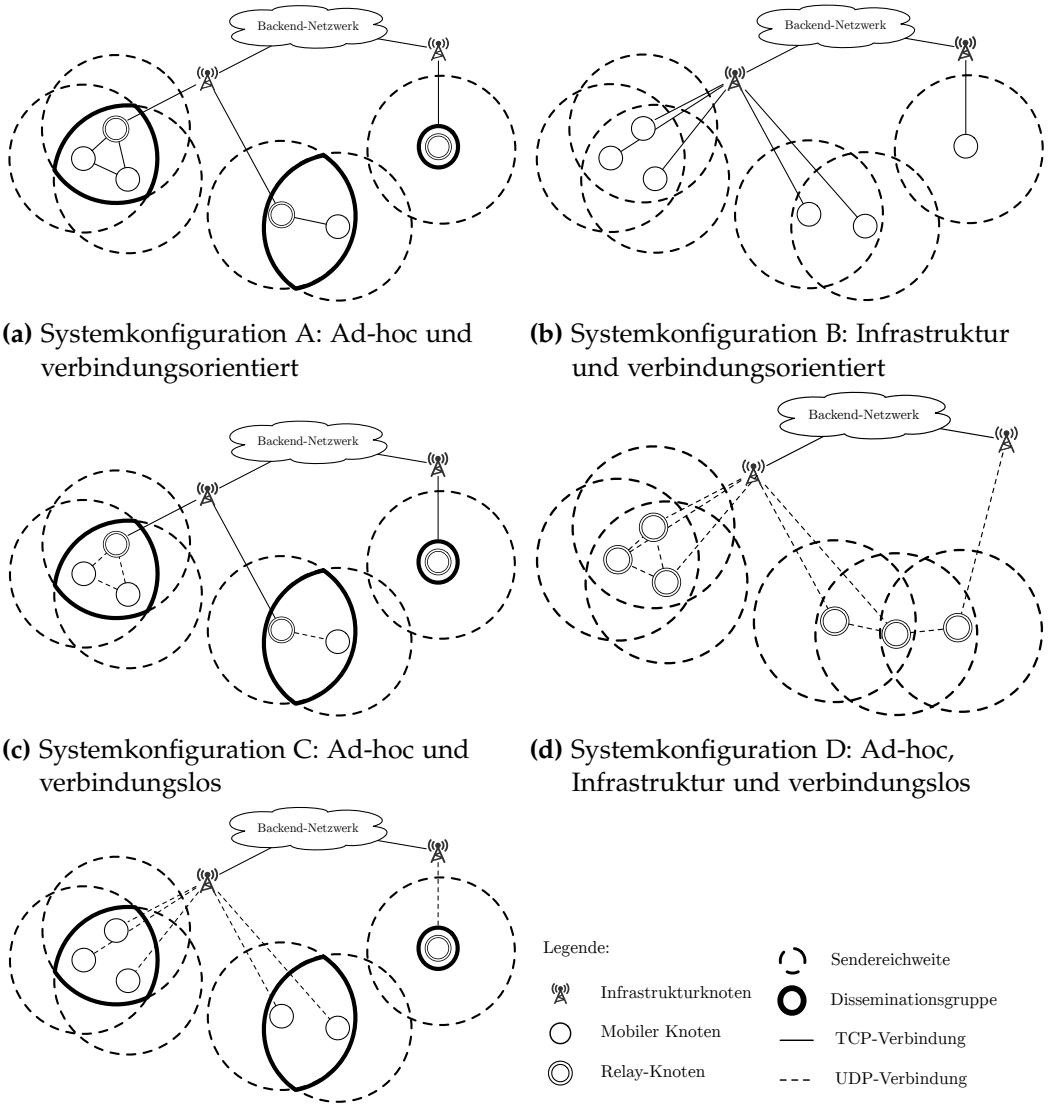
für Kommunikationsanwendungen eingesetzt, in denen niedrige Latenzen wichtig sind (z. B. Video-Streaming-Anwendungen [153]). Falls hingegen die Zuverlässigkeit der Kommunikation im Vordergrund steht, wird das Transportprotokoll TCP gewählt (z. B. beim Nachrichtenaustausch in intelligenten Verkehrssteuerungssystemen [144]).

- Auf der Anwendungsschicht werden verschiedene Kommunikationsmechanismen zur Datenverteilung (engl. data dissemination [123]) als Multimechanismus bereitgestellt. Zur Verteilung der Daten werden mobile Knoten innerhalb sogenannter Disseminationsgruppen organisiert. Kriterien zur Gruppenbildung sind beispielsweise die Entfernung zwischen Knoten, die Zugehörigkeit zum gleichen Ad-hoc-Netzwerk oder eine Kombination verschiedener schichtenübergreifender Kriterien. Innerhalb einer Disseminationsgruppe können Daten über Broadcast an alle Teilnehmer verteilt werden. Alternativ können sogenannte epidemische Protokolle [82] zur Datenverteilung eingesetzt werden, die mit einer konfigurierbaren Wahrscheinlichkeit Datenpakete verwerfen [124].

Durch die Kombination der Kommunikationsmechanismen der verschiedenen Schichten ergeben sich eine Vielzahl konsistenter Systemkonfigurationen. Im Folgenden beschreiben wir exemplarisch einige typische Systemkonfigurationen, in denen sich das betrachtete selbst-adaptive Kommunikationssystem befinden kann.

Beispiel 2.2 (Selbst-adaptives Kommunikationssystem: Systemkonfigurationen) Abbildung 2.1 zeigt schematische Darstellungen der im Folgenden beschriebenen Systemkonfigurationen. Die Platzierung der Knoten deutet deren räumliche Verteilung an.

Systemkonfiguration A: Ad-hoc und verbindungsorientiert
Abbildung 2.1a zeigt das selbst-adaptive Kommunikationssystem, das sich in Systemkonfiguration A befindet. Bei einem selbst-adaptiven Kommunikationssystem, das sich in Systemkonfiguration A befindet, stellen mobile Knoten zueinander Ad-hoc-Kommunikationsverbindungen via Wi-Fi her, falls sie überlappende Sendereichweiten aufweisen. Mobile Knoten, die zueinander Wi-Fi-Verbindungen aufgebaut haben, bilden auf der Anwendungsschicht eine Disseminationsgruppe und nutzen innerhalb der Gruppe TCP als Transportprotokoll. Genau ein Knoten innerhalb der Disseminationsgruppe ist als Relay-Knoten konfiguriert (mit aktivierter Wi-Fi- und LTE-Schnittstelle). Ein Relay-Knoten kommuniziert mit anderen Relay-Knoten über TCP. Die Kommunikation auf physikalischer Ebene erfolgt über den räumlich nächsten Infrastrukturknoten. Hierfür muss die LTE-Schnittstelle in den Relay-Knoten aktiviert sein.



(a) Systemkonfiguration A: Ad-hoc und verbindungsorientiert

(b) Systemkonfiguration B: Infrastruktur und verbindungsorientiert

(c) Systemkonfiguration C: Ad-hoc und verbindungslos

(d) Systemkonfiguration D: Ad-hoc, Infrastruktur und verbindungslos

(e) Systemkonfiguration E: Infrastruktur und verbindungslos

Abbildung 2.1: Exemplarische Systemkonfigurationen zu Beispiel 2.2

Systemkonfiguration B: Infrastruktur und verbindungsorientiert
Abbildung 2.1b zeigt das selbst-adaptive Kommunikationssystem in Systemkonfiguration B. Ein selbst-adaptives Kommunikationssystem in Systemkonfiguration B verfügt über mobile Knoten, die ausschließlich über Infrastrukturknoten über eine aktivierte LTE-Schnittstelle kommunizieren. Hierbei kommt das Transportprotokoll TCP zum Einsatz.

Systemkonfiguration C: Ad-hoc und verbindungslos
Abbildung 2.1c zeigt das selbst-adaptive Kommunikationssystem in Systemkonfiguration C. Bei einem selbst-adaptiven Kommunikationssystem in Systemkonfiguration C stellen mobile Knoten, falls sie überlappende Sendereichweiten aufweisen, zueinander Ad-hoc-Kommunikationsverbindungen via Wi-Fi her und bilden Disseminationsgruppen. Innerhalb der Disseminationsgruppen nutzen mobile Knoten das verbindungslose Transportprotokoll UDP. Genau ein Knoten innerhalb einer Disseminationsgruppe ist als Relay-Knoten (mit aktivierter Wi-Fi- und LTE-Schnittstelle) konfiguriert und kommuniziert mit seinem räumlich nächsten Infrastrukturknoten über LTE mittels TCP.

Systemkonfiguration D: Ad-hoc, Infrastruktur und verbindungslos
Abbildung 2.1d zeigt das selbst-adaptive Kommunikationssystem in Systemkonfiguration D. Bei einem selbst-adaptiven Kommunikationssystem in Systemkonfiguration D, stellen mobile Knoten zu anderen mobilen Knoten in Sendereichweite Ad-hoc-Kommunikationsverbindungen via Wi-Fi her. Sie bilden hierbei auf Anwendungsschicht jedoch keine Disseminationsgruppen. Mobile Knoten nutzen zur Kommunikation über die Wi-Fi-Schnittstelle das Transportprotokoll UDP. Jeder mobile Knoten ist mit genau einem Infrastrukturknoten verbunden und kommuniziert mit diesem ebenfalls mittels UDP.

Systemkonfiguration E: Infrastruktur und verbindungslos
Abbildung 2.1e zeigt das selbst-adaptive Kommunikationssystem, das sich in Systemkonfiguration E befindet. Bei einem selbst-adaptiven Kommunikationssystem, das sich in der genannten Systemkonfiguration befindet, stellen mobile Knoten zueinander Ad-hoc-Kommunikationsverbindungen her. Jeder mobile Knoten ist mit seinem räumlich nächsten Infrastrukturknoten via LTE verbunden und nutzt hierfür das UDP.

Die im Beispiel 2.1 genannten Kommunikationsmechanismen sowie strukturelle Abhängigkeiten zwischen Elementen des Kommunikationssystems können als Architektur bestehend aus Komponenten und Teil-Komponenten spezifiziert werden. Referenzen zwischen Instanzen der Komponenten spezifizieren hierbei logische oder physikalische Abhängigkeiten (z. B. in Form von Kommunikationsverbindungen). Schon bei der Spezifikation der Architektur zur Entwurfszeit muss Variabilität vorgesehen werden, um zur Laufzeit Adaptionen in Form von Komponentenaustausch oder Parameteranpassung innerhalb eines Multimechanismus zu ermöglichen. Hierdurch lassen sich die im

Beispiel 2.2 genannten Systemkonfigurationen realisieren [22]. Zur Strukturierung der externen Adaptionslogik hat sich die MAPE-Referenzarchitektur etabliert [83]. Hierbei werden über den Lebenszyklus einer Adaption hinweg die Phasen (i) Überwachung (engl. monitoring), (ii) Analyse, (iii) Planung und (iv) Ausführung (engl. execution) unterschieden. Die Adaptionslogik stellt die Funktionalität jeder dieser vier Phasen als Komponente bereit. In folgendem Beispiel 2.3 demonstrieren wir die Komponenten der genannten Phasen im betrachteten selbst-adaptiven Kommunikationssystem.

Beispiel 2.3 (Selbst-adaptives Kommunikationssystem: Mechanismen-Adaption)
Monitoring: Die Monitoring-Komponente sammelt fortwährend Umgebungsdaten der verwalteten Ressourcen und identifiziert Symptome, die relevant für das Auslösen einer Adaption sind [72]. Sobald beispielsweise eine Abweichung eines gemessenen Wertes von einem definierten Sollwert identifiziert wird, erkennt die Überwachungskomponente der Adaptionslogik ein für das System relevantes Symptom und gibt die gesammelten Daten an die Analyse-Komponente weiter.

Analyse: Die Analyse-Komponente leitet durch Interferenz aus den Daten eine Änderung der Systemumgebung ab und reicht den identifizierten Systemkontext an die Planungskomponente weiter. Als Parameter der Systemumgebung, die den Kontext eines Systems charakterisieren, werden im betrachteten selbst-adaptiven Kommunikationssystem die Anzahl und die Mobilität der Kommunikationsknoten erfasst. Darüber hinaus kann die Analysekomponente durch eine sprunghafte Veränderung von Umgebungsdaten einen Wechsel des Systemkontexts von einem Normalhin zu einem Notfallbetrieb ableiten.

Planung: Die Planungskomponente berechnet auf Basis des aktuellen Systemkontextes eine konsistente und zielsetzungsadäquate Adaptionsentscheidung. Zur Identifikation konsistenter Systemzustände werden die Architektur, die Konfigurationsoptionen sowie der Kontext des Systems in Form von Modellen erfasst, die zur Entwurfszeit spezifiziert werden. Eine Adaptionsentscheidung repräsentiert eine Systemkonfiguration, bestehend aus einer Menge von Konfigurationsoptionen. Für einen Kontext existiert in der Regel eine Vielzahl konsistenter Systemkonfigurationen, die innerhalb der Planungsphase deshalb zusätzlich durch Qualitätsmetriken (wie z. B. dem erwarteten Einfluss der Adaptionsentscheidung auf die Latenz oder den Energieverbrauch) priorisiert werden [146].

Ausführung: Durch die Ausführungskomponente wird die Rekonfigurationsentscheidung der Planungskomponente in eine Sequenz von Umschaltaktionen umgewandelt, indem an- bzw. abgewählte Konfigurationsoptionen den Komponenten der verwalteten Ressourcen zugeordnet und durch Effektoren umgesetzt werden. In diesem Schritt greift die Adaptionslogik auf Modelle zurück, welche die Zuordnung von Konfigurationsoptionen zu Komponenten der Architektur spezifizieren.

2.2 SOFTWARE-PRODUKTLINIEN

In diesem Abschnitt führen wir die grundlegenden Begriffe und Konzepte von Software-Produktlinien (SPLs) ein.

2.2.1 *Grundlagen*

Die Erfindung von Produktlinien durch Henry Ford Anfang des zwanzigsten Jahrhunderts ermöglichte es, Produkte in großer Stückzahl zu fertigen. Die Massenfertigung in Produktlinien führte durch Produktivitätssteigerungen zu einer enormen Reduktion der Stückkosten gegenüber der bis dahin üblichen individuellen Fertigung. Durch Standardisierung konnte zwar gleichbleibend hohe Produktqualität sichergestellt werden, durch den hieraus resultierenden inflexiblen Fertigungsprozess wurden aber keine kundenindividuellen Anforderungen mehr berücksichtigt.

Bei der Entwicklung von Software ergibt sich eine vergleichbare Problematik: Standard-Software weist in der Regel eine hohe Produktqualität und geringen Wartungsaufwand auf, da die Implementierungsartefakte für alle Kunden gleich sind. Individual-Software ist hingegen kundenspezifisch konzipiert und weist daher in der Regel einen geringen Grad an Wiederverwendbarkeit von Implementierungsartefakten auf, was zu hohen Wartungskosten und geringerer Produktqualität führt. Um dieser Problematik zu begegnen, wird unter dem Begriff kundenindividueller Massenfertigung (engl. mass customization) der Trend zur Kombination von Verfahren der Massen- und individuellen Fertigung zusammengefasst [142]. Software-Produktlinien (SPLs) stellen Technologien und Methoden zur kundenindividuellen Massenfertigung von Software bereit [40]. Bereits in den 1970er Jahren wurden Konzepte zur expliziten Betrachtung von Gemeinsamkeiten verschiedener Software-Varianten mit ähnlicher Funktionalität als Software-Familien vorgeschlagen [108]. In einem strukturierten Entwicklungsprozess werden bei SPLs Gemeinsamkeiten und Unterschiede zwischen (zu entwickelnden) kundenindividuellen Software-Varianten identifiziert, mit dem Ziel, die Wiederverwendbarkeit von Implementierungsartefakten zwischen Software-Varianten zu erhöhen. Durch Modularisierungs- und Plattformkonzepte werden kundenspezifische Implementierungsartefakte von plattformübergreifender Funktionalität separiert und es wird hierdurch die Wiederverwendbarkeit erhöht [112]. Ein (teil-)automatisierter Konfigurations- und Bereitstellungsprozess erlaubt zudem die Kombination von Implementierungsartefakten, um kundenspezifische Produktvarianten abzuleiten [40]. Das Ziel ist es hierbei, durch die Wiederverwendung von Implementierungsartefakten die Produktqualität zu erhöhen und trotzdem individuelle Kundenanforderungen zu erfüllen.

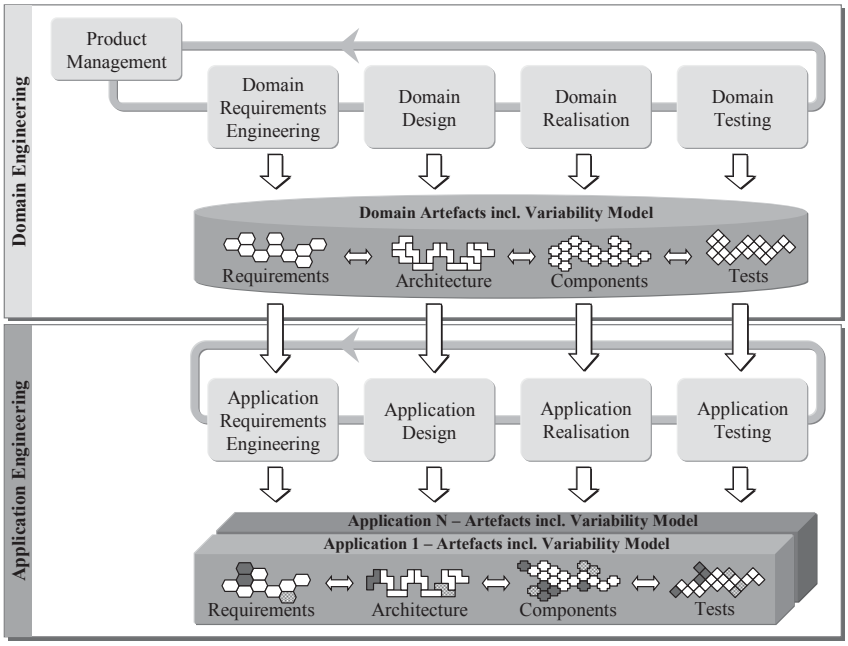


Abbildung 2.2: Entwicklungsprozess von SPLs entnommen aus [112]

Beispiel 2.4 (Selbst-adaptives Kommunikationssystem: Software-Produktlinie) Die Software des adaptiven Kommunikationssystems, das als durchgängiges Beispiel betrachtet wird, kann als SPL konzipiert werden. Wiederverwendbare Komponenten stellen beispielsweise Komponenten dar, die Funktionalität zur Kommunikation über die LTE- oder Wi-Fi-Schnittstelle enthalten. Je nach gefordertem Kommunikationsmechanismus auf physikalischer Schicht, kann eine Software-Variante mit LTE- und/oder Wi-Fi-Komponenten durch einen automatisierten Konfigurationsprozess bereitgestellt werden.

2.2.2 *Entwicklungsprozess bei Software-Produktlinien*

Im Folgenden beschreiben wir den Entwicklungsprozess bei SPLs, der sich wesentlich vom Entwicklungsprozess herkömmlicher Software unterscheidet. Der Entwicklungsprozess bei SPLs ist unterteilt in die Teilprozesse *Domain Engineering* und *Application Engineering* [7]. Abbildung 2.2 zeigt den von Pohl et al. vorgeschlagenen Entwicklungsprozess bei SPLs.

Das Hauptziel des Domain Engineerings ist, Gemeinsamkeiten und Variabilität der SPL zu identifizieren und zu spezifizieren [112]. Es wird definiert, welche Applikationen für die SPL geplant sind und der Umfang der Domäne festgelegt. Darüber hinaus werden die wiederverwendbaren Artefakte spezifiziert und konzipiert, um die gewünsch-

te Variabilität zu erzielen. Anforderungen werden in Form von Variabilitätsmodellen erfasst, Konfigurationsmöglichkeiten der wiederverwendbaren Artefakte werden im Domänen-Entwurf in einem Architekturmodell repräsentiert. Darüber hinaus werden wiederverwendbare Software-Komponenten entworfen und implementiert. Das Ergebnis des Domain Engineerings sind lose gekoppelte Komponenten, die entsprechend der individuellen Kundenanforderungen rekonfigurierbar und kombinierbar sind. Weiterhin ist ein Ergebnis des Domain Engineerings eine Referenzarchitektur, die Schnittstellen zur Kopplung und Austausch konfigurierbarer Komponenten vorsieht.

Das Hauptziel des Application Engineerings ist, die Wiederverwendung von Artefakten aus dem Domain Engineering zu erzielen und variable Systembestandteile entsprechend der Anforderungen der Applikation zu binden [112]. Hierzu wird auf Basis des Variabilitätsmodells aus dem Domain Engineering festgelegt, welche Anforderungen die Applikation erfüllen muss und inwieweit dafür existierende Komponenten der SPL verwendet werden können. Darüber hinaus werden Komponenten entsprechend den Anforderungen mit konfiguriert und an Schnittstellen der Referenzarchitektur gebunden. Das Ergebnis des Application Engineerings ist eine individualisierte Software-Variante bestehend aus einer Architektur mit konfigurierten und an Schnittstellen gebundenen Komponenteninstanzen. Die Architektur der Software-Variante entspricht einer Instanz der Referenzarchitektur der SPL. Der Application-Engineering Teilprozess wird für jede abgeleitete Software-Variante der SPL wiederholt.

Zusätzlich wird beim Entwicklungsprozess von SPLs zwischen den Perspektiven Problemraum und Lösungsraum unterschieden [43]. Der *Problemraum* umfasst alle aus Kundensicht relevanten Prozessschritte und Artefakte. Dies sind insbesondere im Domain Engineering Konfigurationsoptionen, die somit die wichtigste Domänenabstraktion im Problemraum darstellen [7]. Im Application Engineering enthält der Problemraum Kundenanforderungen an eine konkrete Software-Variante und somit konkrete Feature-Selektionen (die Variabilität der SPL binden). Der *Lösungsraum* umfasst alle für Entwickler oder Software-Hersteller relevanten Prozessschritte und Artefakte. Dies sind im Domain Engineering die Referenzarchitektur und rekonfigurierbare Software-Komponenten. Im Application Engineering enthält der Lösungsraum die Architektur (instanziierte Referenzarchitektur) mit konfigurierten Komponenteninstanzen.

2.2.3 *Feature-Modelle*

Zur Erfassung der Variabilität einer SPL werden innerhalb des Domain-Engineerings Variabilitätsmodelle verwendet. Zur Spezifikation von Variabilitätsmodellen haben sich neben Entscheidungsmodellen [133] und orthogonalen Variabilitätsmodellen [112] insbesondere Feature-Modelle [7] etabliert. Bei der *Feature-orientierten Domain Analysis (FODA)* [78] dienen Feature-Modelle der Beschreibung des Konfigurationsraums innerhalb des Domain Engineerings. Bei der Analyse der Problemdomäne zur Identifikation von Gemeinsamkeiten und Unterschieden zwischen Software-Varianten werden *Features*

als grundlegende Domänenabstraktion verwendet [7]. Nach Apel et al. ist ein Feature folgendermaßen definiert:

> „A feature is a characteristic or end-user-visible behavior of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle.“ [7]

Ein Feature ist demnach eine für Nutzer sichtbare Eigenschaft, die geeignet ist, Gemeinsamkeiten und Unterschiede zwischen Software-Varianten zu charakterisieren. Durch die An- und Abwahl von Features wird innerhalb eines strukturierten Konfigurationsprozesses des Application Engineerings eine individuelle Software-Variante abgeleitet. Eine bestimmte Software-Variante korrespondiert somit mit einer Menge von Features, die als Konfiguration bezeichnet wird. Bei einer SPL, deren Problemdomäne aus n Features ohne gegenseitige Abhängigkeiten besteht, können 2^n Konfigurationen und somit 2^n verschiedene Software-Varianten abgeleitet werden. Features weisen in der Regel komplexe Abhängigkeiten zueinander auf. Die Abhängigkeiten schränken die Anzahl zulässiger Konfigurationen ein [43].

Beispiel 2.5 (Features und Abhängigkeiten zwischen Features) Im betrachteten motivierenden Beispiel kann ein mobiler Knoten über die physikalischen Schnittstellen Wi-Fi oder LTE kommunizieren. Beide physikalischen Schnittstellen stellen Features der SPL des adaptiven Kommunikationssystems dar. Es wird zusätzlich vorausgesetzt, dass mindestens eine physikalische Schnittstelle je mobilem Knoten existiert. Die beiden Features `WiFi` und `LTE` weisen somit Abhängigkeiten zueinander auf. Falls ein mobiler Knoten innerhalb einer Disseminationsgruppe als Relay fungiert, müssen sowohl die Features `WiFi` als auch `LTE` ausgewählt sein. Die Software-Variante, die auf dem mobilen Kommunikationsknoten eingesetzt wird, kann somit durch die Feature-Menge $\{\texttt{Relay}, \texttt{LTE}, \texttt{WiFi}\}$ repräsentiert werden. Die Abhängigkeiten werden in Form von Feature-Diagrammen spezifiziert.

Ein Feature-Diagramm ist eine graphische Repräsentation eines Feature-Modells [78]. Abbildung 2.3 zeigt ein Feature-Diagramm, das die Menge gültiger Konfigurationen der SPL des in dieser Arbeit betrachteten Beispiels spezifiziert. Im Folgenden erläutern wir die Syntax und Semantik des gezeigten Feature-Diagramms.

Das Feature-Diagramm strukturiert Features in einer Baumhierarchie, wobei der Wurzel-Knoten per Konvention Bestandteil aller Konfigurationen ist. Innerhalb der Baumstruktur repräsentieren Kanten zwischen Eltern-Features und Kind-Features Kompositionsbeziehungen. Hierbei existieren *verpflichtende* (engl. mandatory) und *optionale* (engl. optional) Beziehungen, die graphisch durch einen ausgefüllten bzw. nicht-ausgefüllten Kreis am Ende der Kante repräsentiert werden: Falls eine Kompositionsbeziehung als verpflichtend spezifiziert ist, muss das Kind-Feature und Eltern-Feature immer gemeinsam

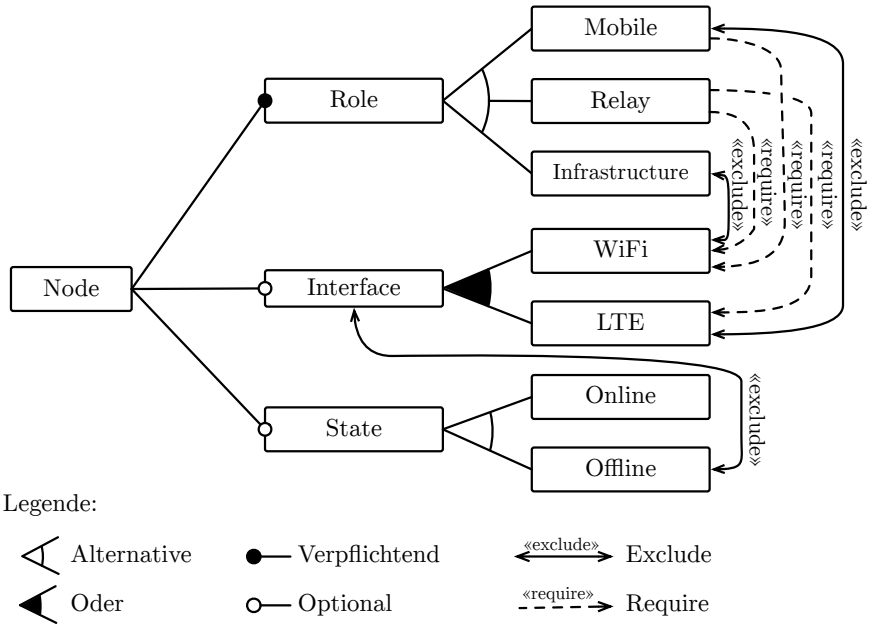


Abbildung 2.3: Feature-Diagramm des motivierenden Beispiels in FODA-Notation

in einer Konfiguration an- oder abgewählt werden. Falls eine Kompositionsbeziehung als optional spezifiziert ist, muss, wenn das Kind-Feature angewählt ist, das Eltern-Feature ebenfalls in der Konfiguration vorkommen (der Umkehrschluss gilt in diesem Fall nicht).

Beispiel 2.6 (Kompositionsbeziehung vom Typ mandatory) In Abbildung 2.3 besteht zwischen dem Kind-Feature `Role` und seinem Eltern-Feature `Node` eine Kompositionsbeziehung vom Typ mandatory. Wenn das Eltern-Feature `Node` in einer Konfiguration gewählt ist, muss somit immer auch das Feature `Role` in der Konfiguration vorkommen und umgekehrt.

Beispiel 2.7 (Kompositionsbeziehung vom Typ optional) In Abbildung 2.3 besteht zwischen dem Kind-Feature `State` und seinem Eltern-Feature `Node` eine optionale Kompositionsbeziehung. Wenn das Eltern-Feature `Node` in einer Konfiguration gewählt ist, kann das Feature `State` somit an- oder abgewählt sein.

Zusätzlich können Abhängigkeiten zwischen Kind-Features des gleichen Eltern-Features durch sogenannte Gruppenbeziehungen spezifiziert werden. Hierbei werden Oder- und Alternativ-Gruppen unterschieden: Innerhalb einer Oder-Gruppe muss mindestens ein Kind-Feature in einer gültigen Konfiguration ausgewählt sein, falls das Eltern-Feature in der Konfiguration ausgewählt ist. Entsprechend muss innerhalb einer Alternativ-

Gruppe genau ein Kind-Feature in der Konfiguration ausgewählt sein, falls das Eltern-Feature in der Konfiguration vorliegt.

Beispiel 2.8 (Oder-Gruppen in Feature-Modellen) In Abbildung 2.3 sind die Kind-Features `WiFi` und `LTE` innerhalb einer Oder-Gruppe und unterhalb des Eltern-Features `Interface` gruppiert. Wenn das Eltern-Feature `Interface` im Konfigurationsprozess ausgewählt wird, muss mindestens `WiFi` oder `LTE` ebenfalls ausgewählt sein. Alternativ dürfen (z. B. im Falle eines Relay-Knotens) auch die Features `WiFi` und `LTE` gleichzeitig in der Konfiguration vorkommen, falls das Eltern-Feature `Interface` ausgewählt ist.

Beispiel 2.9 (Alternativ-Gruppen in Feature-Modellen) In Abbildung 2.3 sind die Kind-Features `Mobile`, `Relay`, `Infrastructure` in einer Alternativ-Gruppe unterhalb des Eltern-Features `Role` gruppiert. Da die Kind-Features in einer Alternativ-Beziehung zueinander stehen, muss immer genau eines der Kind-Features in einer Konfiguration vorkommen, falls das Feature `Role` ausgewählt ist.

Weiterhin werden im Feature-Diagramm Abhängigkeiten zwischen beliebigen Features über die Baumstruktur hinweg mittels sogenannter *Cross-Tree-Constraints* spezifiziert. Hierbei werden die Constraint-Typen *Require* und *Exclude* unterschieden, die im Feature-Diagramm als gerichtete bzw. ungerichtete Kante zwischen Start- und End-Feature repräsentiert werden: Falls das Start-Feature einer Require-Abhängigkeit ausgewählt ist, muss auch das End-Feature ausgewählt sein. Bei einer Exclude-Abhängigkeit dürfen Start- und End-Features nicht gleichzeitig ausgewählt sein.

Beispiel 2.10 (Require-Constraints in Feature-Modellen) In Abbildung 2.3 besteht zwischen den Features `Relay` und `WiFi` eine Require-Constraint, wobei das Feature `Relay` als Start- und das Feature `WiFi` als End-Feature spezifiziert ist. In einer Konfiguration darf das Feature `Relay` somit nur vorliegen, falls gleichzeitig das Feature `WiFi` ausgewählt ist.

Beispiel 2.11 (Exclude-Constraints in Feature-Modellen) In Abbildung 2.3 besteht zwischen den Features `Mobile` und `LTE` eine Exclude-Constraint. In einer Konfiguration darf das Feature `Mobile` somit nur vorliegen, falls nicht gleichzeitig das Feature `LTE` gewählt ist und umgekehrt.

Die zuvor anhand der Syntax beschriebene Konfigurationssemantik von Feature-Modellen kann als aussagenlogische Formel codiert werden [16]. Ein Feature wird hierbei durch eine binäre Variable repräsentiert und durch aussagenlogische Verknüpfungen zu anderen Variablen, die ebenfalls Features repräsentieren, in Beziehung gesetzt. Die

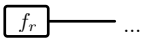
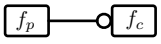
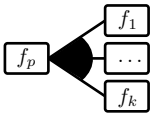
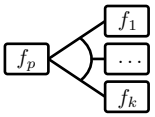
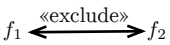
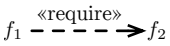
Element	Syntax im Feature-Diagramm	Aussagenlogische Repräsentation im Feature-Modell
Wurzel		f_r
Optional		$f_c \implies f_p$
Verpflichtend		$f_c \iff f_p$
Oder-Gruppe		$f_p \iff (f_1 \vee \ldots \vee f_k)$
Alternativ-Gruppe		$f_p \iff (f_1 \wedge \neg(\ldots \vee f_k) \vee \ldots \vee \neg(f_1 \vee \ldots) \wedge f_k)$
Exclude		$\neg(f_1 \wedge f_2)$
Require		$f_1 \implies f_2$

Tabelle 2.1: Aussagenlogische Repräsentation von Elementen des Feature-Diagramms

Tabelle 2.1 enthält für syntaktische Elemente von Feature-Diagrammen die aussagenlogischen Korrespondenzen. Die Semantik eines Feature-Modells entspricht der konjunktiven Verknüpfung der durch die Korrespondenzen übersetzten syntaktischen Elemente.

Eine Variablenbelegung, die dazu führt, dass die resultierende aussagenlogische Formel erfüllbar ist, repräsentiert eine konsistente Konfiguration. Falls eine Variable innerhalb einer erfüllbaren Variablenbelegung den Wert *Wahr* aufweist, liegt das durch die Variable codierte Feature in der Konfiguration vor. Falls eine Variable den Wert *Falsch* aufweist, ist das durch die Variable codierte Feature in der Konfiguration hingegen nicht ausgewählt. Alle zulässigen Konfigurationen eines Feature-Modells können identifiziert werden, indem alle erfüllbaren Variablenzuweisungen der aussagenlogischen Repräsentation gefunden werden [96, 97]. Variablenbelegungen, die zu nicht erfüllbaren aussagenlogischen Formeln führen, entsprechen unzulässigen und damit inkonsistenten Konfigurationen.

Konfiguration	Node	Role	Mobile	Relay	Infrastructure	Interface	WiFi	LTE	State	Online	Offline
1	W	W	W	F	W	W	W	F	W	W	F
2	W	W	W	F	W	W	W	F	F	F	F
3	W	W	F	W	F	W	F	W	W	W	F
4	W	W	F	W	F	W	F	W	F	F	F
5	W	W	F	F	W	W	F	W	W	W	F
6	W	W	F	F	W	W	F	W	F	F	F

Tabelle 2.2: Zulässige Konfigurationen des Feature-Modells in Abbildung 2.3

Beispiel 2.12 (Aussagenlogische Repräsentation von Feature-Modellen) Das in Abbildung 2.3 gezeigte Feature-Diagramm kann anhand der Korrespondenzen in folgende aussagenlogische Formel übersetzt werden:

$$\begin{aligned}
&\texttt{Knoten} \wedge (\texttt{Knoten} \iff \texttt{Role}) \wedge (\texttt{Interface} \implies \texttt{Node}) \wedge \\
&(\texttt{Role} \iff (\texttt{Mobile} \wedge \neg\texttt{Relay} \wedge \neg\texttt{Infrastructure} \vee \\
&\neg\texttt{Mobile} \wedge \texttt{Relay} \wedge \neg\texttt{Infrastructure} \vee \\
&\neg\texttt{Mobile} \wedge \neg\texttt{Relay} \wedge \texttt{Infrastructure})) \wedge \\
&(\texttt{State} \implies \texttt{Node}) \wedge (\texttt{Interface} \iff (\texttt{WiFi} \vee \texttt{LTE})) \wedge \\
&(\texttt{State} \iff (\texttt{Online} \wedge \neg\texttt{Offline}) \vee (\neg\texttt{Online} \wedge \texttt{Offline})) \wedge \\
&\neg(\texttt{Mobile} \wedge \texttt{LTE}) \wedge (\texttt{Relay} \implies \texttt{LTE}) \wedge (\texttt{Relay} \implies \texttt{WiFi}) \wedge \\
&(\texttt{Mobile} \implies \texttt{Wifi}) \wedge \neg(\texttt{Infrastructure} \wedge \texttt{WiFi}) \wedge \neg(\texttt{Interface} \wedge \texttt{Offline})
\end{aligned}$$

Ein geklammerter Term der Formel korrespondiert zu einem Term aus Tabelle 2.1.

Tabelle 2.2 enthält alle erfüllbaren Variablenzuweisungen der aussagenlogische Formel, die das Feature-Diagramm aus Abbildung 2.3 repräsentiert. Insgesamt existieren sechs Variablenbelegungen, die dazu führen, dass die aussagenlogische Formel erfüllbar ist. Der Konfigurationsraum, der durch das Feature-Diagramm spezifiziert wird, umfasst dementsprechend sechs Konfigurationen.

2.2.4 *Erweiterte Feature-Modelle*

In dem im Rahmen dieser Arbeit betrachteten Beispiel können Elemente des Problemraums potentiell mehrfach instanziiert werden (z. B. enthält das selbst-adaptive Kommunikationssystem mehrere Instanzen verteilter Knoten, die individuell konfiguriert sind). Darüber hinaus treten numerische Konfigurationsparameter auf (z. B. die probabilistische Verwurfrate von Datenpaketen eines epidemischen Disseminationsprotokolls). Konfigurationsparameter können wiederum potentiell Abhängigkeiten zu (binären) Konfigurationsoptionen oder anderen (numerischen) Konfigurationsparametern haben. Um derartige Problemstellungen zu beschreiben, werden in der Literatur gegenüber den zuvor vorgestellten Feature-Modellen zwei Erweiterungen der Konfigurationssemantik vorgeschlagen, welche die Ausdrucksstärke der Spezifikation erhöhen: (i) nicht-binäre Feature-Typen [43, 78] und (ii) Multiplizitätsintervalle [125]. Nicht-binäre Feature-Typen können in *attributierten Feature-Modellen* numerische Konfigurationsparameter repräsentieren, Multiplizitätsintervalle erlauben in *kardinalitätsbasierten Feature-Modellen* (engl. Cardinality-Based Feature Models (CFM)) die Spezifikation mehrerer Instanzen eines Feature-Typs in einer Konfiguration. Beide genannten Erweiterungen können auch in einer Spezifikation kombiniert werden, was jedoch zu einer komplexeren Semantik führt und damit die Analyse derartiger Spezifikationen herausfordernd macht (siehe Herausforderung 2 in Abschnitt 1.1). Im Folgenden beschreiben wir Syntax und Semantik von attributierten und kardinalitätsbasierten Feature-Modellen, die im Rahmen dieser Arbeit kombiniert zum Einsatz kommen, um den Problemraum des betrachteten selbst-adaptiven Kommunikationssystems zu spezifizieren.

Attributierte Feature-Modelle

Attributierte Feature-Modelle erweitern die Konfigurationssemantik gegenüber herkömmlichen Feature-Modellen um nicht-binäre Feature-Typen. Ein numerisches Feature (in der Literatur synonym auch als Feature-Attribut bezeichnet) repräsentiert eine messbare Charakteristik des Problemraums, die komplexe Abhängigkeiten zu binären Features oder anderen numerischen Features aufweisen kann [43]. Kang et al. erwähnten numerische Features bereits als mögliche Erweiterung gegenüber herkömmlichen Feature-Modellen in ihren frühen Arbeiten [78] und schlugen numerische Features zudem zur Repräsentation nicht-funktionaler Eigenschaften vor [79]. Numerische Features weisen immer eine Domäne und einen Wertebereich auf [80]. Im Rahmen dieser Arbeit betrachten wir ganzzahlige und reellwertige Domänen mit potentiell unbeschränkten Wertebereichen.

Abbildung 2.4 zeigt ein erweitertes Feature-Modell mit numerischen Features. Numerische Features werden unterhalb von Features in der Baumhierarchie angeordnet, können jedoch selbst nicht über eigene Kind-Features verfügen. Falls ein numerisches Feature in der Konfiguration vorliegt, muss sein Eltern-Feature ebenfalls in der Konfiguration ausgewählt sein.

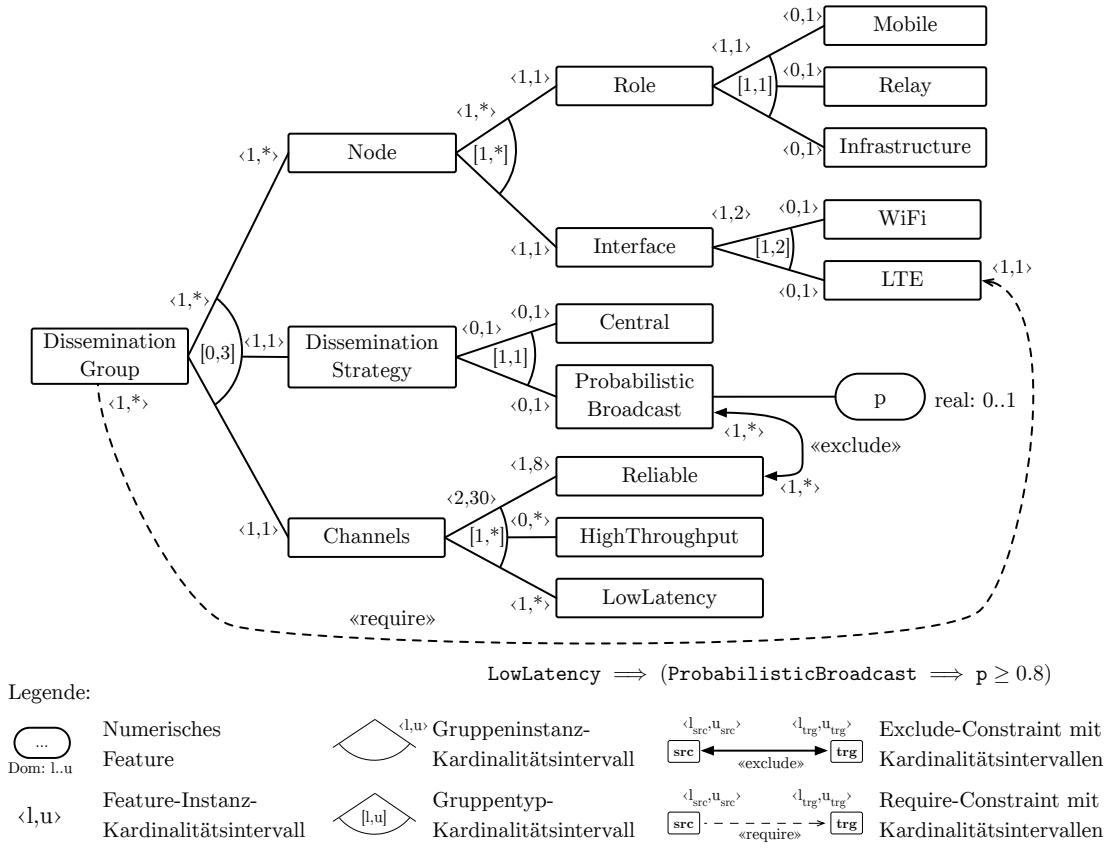


Abbildung 2.4: Erweitertes kardinalitätsbasiertes Feature-Diagramm des betrachteten Beispiels

In der graphischen Syntax werden numerische Features als abgerundete Rechtecke dargestellt, die mit einem Schlüsselwort zum Kennzeichnen der Domäne sowie einem Intervall zur Definition des Wertebereichs annotiert sind. Ganzzahlige numerische Features werden in der graphischen Syntax durch das Schlüsselwort `integer` und reellwertige numerische Features werden durch das Schlüsselwort `real` gekennzeichnet. Der Wertebereich wird durch die Notation $l..u$ angegeben, wobei l der unteren und u der oberen Grenze des Wertebereichs entspricht. Falls es keine Schranken für l und u gibt, wird das spezielle Symbol $*$ verwendet. Die Möglichkeit, potentiell unbeschränkte Wertebereiche zu spezifizieren, stellt eine Erweiterung gegenüber existierenden Sprachen zur Spezifikation attributierter Feature-Modelle dar [80, 81, 110] und wurde von Weckesser et al. in [149] für kardinalitätsbasierte Feature-Modelle vorgestellt.

Beispiel 2.13 (Attributiertes Feature-Modell) Abbildung 2.4 zeigt ein erweitertes Feature-Modell mit dem numerischen Feature `p`, das die Verwurfrate eines epidemischen Disseminationsprotokolls angibt. Das numerische Feature `p` verfügt über einen reellen Wertebereich, der durch das Intervall 0..1 eingeschränkt ist. Konfigurationen können somit numerische Features `p` mit Werten zwischen 0 und 1 enthalten. Falls das numerische Feature in einer Konfiguration auftritt, muss ebenfalls das Eltern-Feature `ProbabilisticBroadcast` ausgewählt sein.

Zusätzlich können numerische Features komplexe Abhängigkeiten zu anderen binären oder numerischen Features aufweisen. Zur Spezifikation solcher Abhängigkeiten verwenden wir Constraints, die durch aussagenlogische Formeln und arithmetische Vergleichsoperatoren numerische und binäre Features verknüpfen.

Beispiel 2.14 (Abhängigkeiten zwischen Features und Attributen in erweiterten Feature-Modellen) Im Beispiel, das in Abbildung 2.4 zu sehen ist, besteht eine komplexe Abhängigkeit zwischen dem numerischen Feature `p` und dem binären Feature `LowLatency`: Falls das Feature `LowLatency` in einer Konfiguration vorliegt, muss der Wert des numerischen Features `p` mindestens gleich 0.8 sein, falls sein Eltern-Feature `ProbabilisticBroadcast` ebenfalls in der Konfiguration auftritt. Die beschriebene Abhängigkeit wird durch folgenden Constraint spezifiziert:

$$\texttt{LowLatency} \implies (\texttt{ProbabilisticBroadcast} \implies \texttt{p} \geq 0.8).$$

Kardinalitätsbasierte Feature-Modelle

Kardinalitätsbasierte Feature-Modelle erweitern die Konfigurationssemantik gegenüber herkömmlichen Feature-Modellen um die Multiinstanziierung von Feature-Typen. Riebisch et al. schlagen in [125] erstmals die Erweiterung von Feature-Modellen um Multiplizitäten vor, wie sie beispielsweise in UML-Klassendiagrammen üblich sind, um die Charakterisierung mehrerer Instanzen eines Feature-Typs in einer Konfiguration zu ermöglichen. Czarnecki et al. schlagen in [44] Kardinalitätsintervalle für Features und Gruppen vor und systematisieren in [45] die Erweiterung der Ausdrucksstärke als kardinalitätsbasierte Feature-Modelle. Abbildung 2.4 zeigt ein erweitertes kardinalitätsbasiertes Feature-Modell mit verschiedenen Arten von Kardinalitätsintervallen, deren Syntax und Semantik im Folgenden erläutert werden. Die Syntax und Semantik basiert auf der von Weckesser et al. in [149] eingeführten Repräsentation.

Feature-Instanz-Kardinalitätsintervalle wurden ursprünglich von Czarnecki et al. in [44] als sogenannte Feature-Kardinalitäten eingeführt. Ein Feature-Instanz-Kardinalitätsintervall ist in der graphischen Syntax auf der linken Seite eines Feature-Typs durch ein Intervall der Form $\langle l, u \rangle$ gekennzeichnet. Das Intervall legt die minimal und maximal erlaubte Anzahl von Feature-Instanzen pro Eltern-Feature-Instanz in einer Konfiguration

fest. Das spezielle Symbol $*$ wird als obere Schranke u verwendet, falls das Intervall nach oben hin unbeschränkt ist.

Beispiel 2.15 (Feature-Instanz-Kardinalitätsintervalle) Im Beispiel, das in Abbildung 2.4 gezeigt ist, ist der Feature-Typ `Reliable` mit dem Feature-Instanz-Kardinalitätsintervall $\langle 1, 8 \rangle$ annotiert. In einer Konfiguration müssen somit je Instanz des Eltern-Feature-Typs `Channel` mindestens eine und höchstens acht Instanzen des Feature-Typs `Reliable` vorliegen.

Gruppeninstanz-Kardinalitätsintervalle wurden von Czarnecki et al. in [44] als sogenannte Gruppen-Kardinalitäten eingeführt. Ein Gruppeninstanz-Kardinalitätsintervall ist in der graphischen Syntax auf der rechten Seite eines Feature-Typs durch ein Intervall der Form $\langle l, u \rangle$ gekennzeichnet. Das Intervall legt die minimal und maximal erlaubte Anzahl der Summe der Kind-Feature-Instanzen einer Eltern-Feature-Instanz fest. Wie zuvor gibt das Symbol $*$ an, dass ein Intervall nach oben hin unbeschränkt ist.

Beispiel 2.16 (Gruppeninstanz-Kardinalitätsintervalle) Im Beispiel in Abbildung 2.4 ist der Eltern-Feature-Typ `Channels` mit dem Gruppeninstanz-Kardinalitätsintervall $\langle 2, 30 \rangle$ annotiert. In einer Konfiguration muss somit pro Eltern-Feature-Instanz `Channels` die Anzahl der Kinder-Feature-Instanzen der Feature-Typen `Reliable`, `HighThroughput` und `LowLatency` mindestens gleich zwei jedoch höchstens 30 sein. Das Intervall beschränkt die minimale und maximale Anzahl von Verbindungskanälen je Disseminationsgruppe. Gruppeninstanz-Kardinalitätsintervalle sind daher insbesondere geeignet, um Ressourcenbeschränkungen zu spezifizieren.

Ein *Gruppentyp-Kardinalitätsintervall* ist in der graphischen Syntax auf der rechten Seite eines Feature-Typs durch ein Intervall der Form $[l, u]$ gekennzeichnet. Das Intervall legt die minimal und maximal erlaubte Anzahl ausgewählter Kind-Feature-Typen je Instanz eines Eltern-Feature-Typs fest. Im Gegensatz zu den bisher vorgestellten Kardinalitätsintervallen schränken Gruppentyp-Kardinalitätsintervalle Feature-Typen und nicht Feature-Instanzen ein: Ein Kind-Feature-Typ einer Instanz eines Eltern-Feature-Typs liegt in einer Konfiguration vor, falls mindestens eine Instanz des Kind-Feature-Typs pro Eltern-Feature-Instanz gewählt ist. Wie bei anderen Arten von Kardinalitätsintervallen kennzeichnet das Symbol $*$ unbeschränkte Gruppentyp-Kardinalitätsintervalle.

Beispiel 2.17 (Gruppentyp-Kardinalitätsintervalle) Im Beispiel, das in Abbildung 2.4 gezeigt ist, ist der Eltern-Feature-Typ `Interface` mit dem Gruppentyp-Kardinalitätsintervall $[1, 2]$ annotiert. In einer Konfiguration müssen somit pro Instanz des Feature-Typs `Interface` mindestens ein und höchstens zwei Kind-Feature-Typen `WiFi` und `LTE` gewählt sein. Es sind zwei Kind-Feature-Typen in einer Konfiguration gewählt,

falls pro Instanz des Feature-Typs `Interface` jeweils mindestens eine Instanz der Feature-Typen `WiFi` und `LTE` vorkommt.

Darüber hinaus können in kardinalitätsbasierten Feature-Modellen Cross-Tree-Constraints mit Kardinalitätsintervallen annotiert werden [119, 149]. In der graphischen Syntax wird hierzu ein Quell-Kardinalitätsintervall der Form $\langle l_{\text{src}}, u_{\text{src}} \rangle$ am Start und ein Ziel-Kardinalitätsintervall der Form $\langle l_{\text{trg}}, u_{\text{trg}} \rangle$ am Ende einer Cross-Tree-Constraint-Kante annotiert. Die entsprechend annotierten Feature-Typen werden als Quell- und Ziel-Feature-Typen bezeichnet. Im Folgenden beschreiben wir die Semantik des annotierten Kardinalitätsintervalls für Require- und Exclude-Constraints.

Für einen gerichteten *Require-Constraint* gilt, falls in einer Konfiguration mindestens l_{src} und höchstens u_{src} Instanzen des Quell-Feature-Typs vorliegen, dass mindestens l_{trg} und höchstens u_{trg} Instanzen des Ziel-Features gewählt sein müssen. Falls die Anzahl von Instanzen des Quell-Feature-Typs nicht innerhalb der Intervallgrenzen des Quell-Kardinalitätsintervalls liegt, hat der Require-Constraint keine Gültigkeit.

Beispiel 2.18 (Require-Constraints in kardinalitätsbasierten Feature-Modellen) Im Beispiel, das in Abbildung 2.4 gezeigt ist, existiert ein Require-Constraint zwischen dem Quell-Feature `DisseminationGroup` und dem Ziel-Feature `LTE`. Am Quell-Feature ist das Quell-Kardinalitätsintervall $\langle 1, * \rangle$ annotiert. Entsprechend ist am Ziel-Feature das Ziel-Kardinalitätsintervall $\langle 1, 1 \rangle$ annotiert. Falls somit der Feature-Typ `DisseminationGroup` mindestens eine Instanz in der Konfiguration aufweist, muss genau eine Instanz des Feature-Typs `LTE` in der Konfiguration vorliegen.

Für einen ungerichteten *Exclude-Constraint* gilt, falls in einer Konfiguration mindestens l_{src} und höchstens u_{src} Instanzen des Quell-Feature-Typs vorliegen, dass die Konfiguration weniger als l_{trg} oder mehr als u_{trg} Instanzen des Ziel-Feature-Typs aufweisen muss. Umgekehrt, falls vom Ziel-Feature-Typ in einer Konfiguration mindestens l_{trg} und höchstens u_{trg} Instanzen vorliegen, muss die Konfiguration weniger als l_{src} oder mehr als u_{src} Instanzen des Quell-Feature-Typs aufweisen.

Beispiel 2.19 (Exclude-Constraints in kardinalitätsbasierten Feature-Modellen) Im Beispiel, das in Abbildung 2.4 gezeigt ist, existiert eine Exclude-Constraint zwischen dem Quell-Feature `ProbabilisticBroadcast` mit dem Quell-Kardinalitätsintervall $\langle 1, * \rangle$ und dem Ziel-Feature `Reliable` mit dem Ziel-Kardinalitätsintervall $\langle 1, * \rangle$. Falls somit eine Konfiguration mindestens eine Instanz des Quell-Feature-Typs `ProbabilisticBroadcast` aufweist, dürfen nicht gleichzeitig Instanzen des Ziel-Feature-Typs `Reliable` auftreten. Umgekehrt dürfen, falls in einer Konfiguration mindestens eine Instanz des Ziel-Feature-Typs `Reliable` auftritt, nicht gleichzeitig Instanzen von `ProbabilisticBroadcast` vorkommen.

Michel et al. weisen in [105] auf Widersprüchlichkeiten bei der Interpretation von Cross-Tree-Constraints in kardinalitätsbasierten Feature-Modellen hin: Ein Cross-Tree-Constraint kann sich entweder auf Feature-Typen oder auf Feature-Instanzen beziehen. Falls sich ein Cross-Tree-Constraint auf Feature-Typen bezieht, entspricht dies einer *globalen* Interpretation, da Aussagen über alle Instanzen eines Feature-Typs getroffen werden. Bezieht sich hingegen ein Cross-Tree-Constraint auf (einzelne) Feature-Instanzen, entspricht dies einer *lokalen* Interpretation (da sie eine Aussage über die minimale und maximale Anzahl von Feature-Instanzen je Gruppen treffen).

Um sowohl eine globale als auch eine lokale Interpretation innerhalb von Dekompositionsbeziehungen zwischen Features zu ermöglichen, wurden zuvor Gruppeninstanz- und Gruppentyp-Kardinalitätsintervalle unterschieden. Feature-Instanz-Kardinalitätsintervalle entsprechen hingegen immer einer lokalen Interpretation. Für Cross-Tree-Constraints in kardinalitätsbasierten Feature-Modellen und numerische Features wählen wir im Folgenden für kardinalitätsbasierte Feature-Modelle eine globale Interpretation, da dies der ursprünglichen anforderungsorientierten Problemraumbetrachtung am nächsten kommt [149].

Die Konfigurationssemantik kardinalitätsbasierter Feature-Modelle unterscheidet sich gegenüber herkömmlichen Feature-Modellen, die in Abschnitt 2.2.3 vorgestellt wurden: Bei kardinalitätsbasierten Feature-Modellen dürfen in einer Konfiguration Kind-Feature-Typen je Instanz ihres Eltern-Feature-Typs mehrfach instanziiert werden, soweit keine Bedingungen der Kardinalitätsintervalle verletzt werden. Eine Instanz eines Kind-Feature-Typs ist innerhalb einer Konfiguration somit abhängig von genau einer Instanz seines Eltern-Feature-Typs. Da wir für numerische Features innerhalb von kardinalitätsbasierten Feature-Modellen eine globale Interpretation voraussetzen, gilt die Annahme, dass alle Instanzen eines numerischen Features den gleichen Wert aufweisen und Einschränkungen immer für alle Instanzen eines numerischen Features gelten. Durch die mehrfache Instanziierung eines Kind-Feature-Typs je Eltern-Feature-Typ resultieren sogenannte geklonte Teilbäume, die aus replizierten Feature-Instanzen bestehen [44]. Da Kardinalitätsintervalle durch das Symbol $*$ als unbeschränkt spezifiziert werden können, ergeben sich potentiell unendlich viele geklonte Teilbäume von Feature-Instanzen. Folgendes Beispiel demonstriert die Konfigurationssemantik von erweiterten kardinalitätsbasierten Feature-Modellen.

Beispiel 2.20 (Konfiguration des erweiterten kardinalitätsbasierten Feature-Modells) Abbildung 2.5 zeigt eine beispielhaften Konfiguration des erweiterten kardinalitätsbasierten Feature-Modells aus Abbildung 2.4. Die dargestellte Konfiguration entspricht der Systemkonfiguration C aus Abbildung 2.1c: Drei Instanzen des Feature-Typs `Node` existieren, wobei jeweils ein Knoten als `Relay` und `Mobile` konfiguriert ist. Zusätzlich ist der Disseminationsgruppe ein Infrastrukturknoten zugeordnet. In der Disseminationsgruppe existieren zwei Instanzen des Feature-Typs `LowLatency`, was zu UDP-Kommunikationsverbindungen zwischen den mobilen Knoten im Lö-

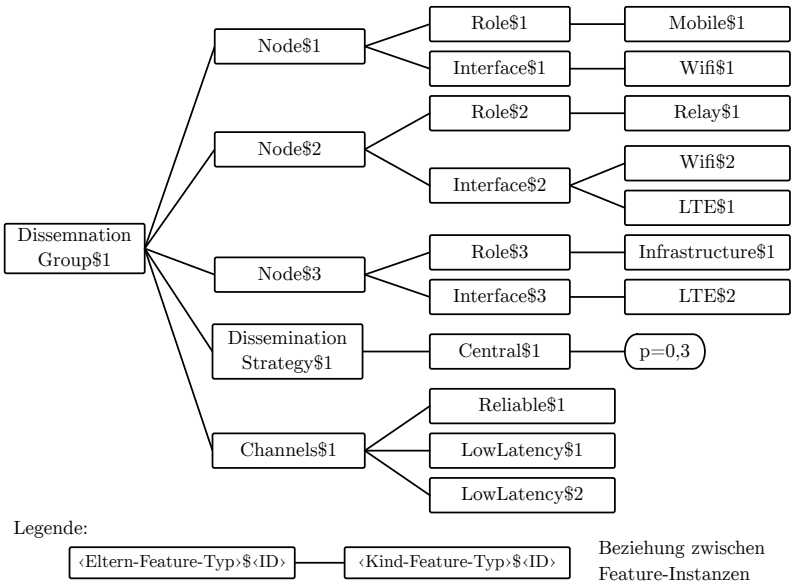


Abbildung 2.5: Konfiguration des erweiterten kardinalitätsbasierten Feature-Modells in Abbildung 2.4

sungsraum korrespondiert, und eine Instanz des Feature-Typs `Reliable`, was zu einer TCP-Kommunikationsverbindung zwischen Relay- und Infrastrukturknoten im Lösungsraum korrespondiert. Da der Feature-Typ `Node` mit dem Feature-Instanz-Kardinalitätsintervall $\langle 1, * \rangle$ annotiert ist, können potentiell beliebig viele Instanzen dieses Feature-Typs in einer Konfiguration auftreten. Da das Feature `Role` Kind-Feature-Typ des Eltern-Feature-Typs `Node` ist und je Instanz des Feature-Typs `Node` genau ein mal instanziiert sein muss, kann der Feature-Typ `Role` ebenfalls potentiell beliebig viele Instanzen in einer Konfiguration aufweisen.

2.2.5 *Analyse von Feature-Modellen*

Die Analyse von Feature-Modellen dient dazu, Informationen aus der Spezifikation eines Feature-Modells zu extrahieren, um Inkonsistenzen, Anomalien oder sonstige Angaben zur Beschaffenheit des Problemraums zu identifizieren [17]. Die Analyse von Feature-Modellen erfordern automatisierte Verfahren, da sie fehleranfällig, aufwändig und aufgrund der Spezifikationsgrößen nicht manuell durchführbar sind [20]. Zur Analyse von Feature-Modellen unterscheiden wir aussagenlogische und algebraische Ansätze: Bei aussagenlogischen Ansätzen werden die Semantik von Feature-Modellen und das zugehörige Analyseziel als Erfüllbarkeitsproblem (engl. Propositional Satisfiability Problem (SAT)) [16, 93, 102] oder Constraint-Erfüllbarkeitsproblem (engl. Constraint Satisfaction Problem (CSP)) [19, 21] codiert und mittels Standard-Lösungsverfahren ge-

löst. Dies entspricht der in Beispiel 2.12 demonstrierten Konfigurationssemantik. Bei den von Schobbens et al. in [135] und Heymans et al. in [69] präsentierten algebraischen Ansätzen werden die Konfigurationssemantik und das zugehörige Analyseziel mittels mathematischer Gleichungen und Mengen codiert, sodass Analyseziele analytisch charakterisiert werden können. Im Folgenden betrachten wir aussagenlogische Ansätze, die eine automatisierte Analyse von Feature-Modellen mittels Standard-Lösungsverfahren ermöglichen.

Für die Charakterisierung der Konfigurationssemantik eines Feature-Modells und des zugehörigen Analyseziels als Erfüllbarkeits- und Constraint-Erfüllbarkeitsproblem werden Spezifikationselemente in aussagenlogische Formeln übersetzt: Bei herkömmlichen Feature-Modellen wird für jedes Feature eine Variable eingeführt und durch aussagenlogische Formeln mit den Variablen anderer Features verknüpft (siehe Beispiel 2.12). Für numerische Features in attributierten Feature-Modellen werden, falls die Konfigurationssemantik als Erfüllbarkeitsproblem charakterisiert wird, mögliche Belegungen des Wertebereichs als binäre Variablen codiert [29]. Falls die Konfigurationssemantik als Constraint-Erfüllbarkeitsproblem charakterisiert wird, wird der Wertebereich eines numerischen Features durch Constraints eingeschränkt [19, 80]. In beiden Fällen wird vorausgesetzt, dass die Wertebereiche beschränkt und im Vorhinein bekannt sind.

Für die Charakterisierung der Konfigurationssemantik kardinalitätsbasierter Feature-Modelle als Constraint-Erfüllbarkeitsproblem [41, 117, 119] oder Binary Decision Diagram (BDD) [159] werden Feature-Instanzen als binäre Variablen codiert. Hierbei wird ebenfalls vorausgesetzt, dass Kardinalitätsintervalle beschränkt sind. Es ist daher nicht möglich, unbeschränkte Wertebereiche von numerischen Features in attributierten Feature-Modellen und unbeschränkter Kardinalitätsintervalle in kardinalitätsbasierten Feature-Modellen zu analysieren (siehe Herausforderung 4). Zudem führt die beschriebene Codierung bei großen Wertebereichen oder reellwertigen numerischen Features zu Effizienzproblemen (siehe Herausforderung 3). Im Folgenden stellen wir Validierungs- und Analyseziele vor, die auf die zuvor vorgestellten Problemraumspezifikationen angewendet werden können.

Ein wesentliches Analyseziel ist die Prüfung von Spezifikationen von Feature-Modellen auf *Konsistenz*. Ein Feature-Modell ist konsistent, falls es mindestens eine zulässige Konfiguration besitzt.

> **Beispiel 2.21** (Konsistenzprüfung von Feature-Modellen) Das in Abbildung 2.3 gezeigte Feature-Modell lässt sich, wie in Beispiel 2.12 demonstriert, als aussagenlogische Formel repräsentieren. Für die aussagenlogische Formel können sechs zulässige Variablenbelegungen gefunden werden. Für das Feature-Modell existieren somit sechs zulässige Konfigurationen. Das betrachtete Feature-Modell ist daher konsistent.

Ein weiteres wichtiges Analyseziel ist die Validierung von *Konfigurationen* [20, 151]. Bei der Validierung wird eine Konfiguration, die eine Software-Variante einer SPL re-

präsentiert, daraufhin überprüft, ob sie Teil des Konfigurationsraums ist. Die Analyse kann sowohl für vollständige als auch partielle Konfigurationen durchgeführt werden. Eine partielle Konfiguration enthält nur für eine Teilmenge aller Features eines Feature-Modells Angaben, ob diese an- oder abgewählt sind.

Darüber hinaus ist die Erkennung von Anomalien in Feature-Modellen ein wichtiges Analyseziel. Eine Software-Anomalie ist im IEEE-Standard 1044-2009 definiert als Irregularität, Inkonsistenz oder sonstige Abweichung von Erwartungen hinsichtlich Verhalten, Form oder Funktion [74]. Benavides et al. systematisieren folgende Anomalietypen, die in Feature-Modellen auftreten können [20]:

- Eine Anomalie vom Typ *totes Feature* (engl. dead feature) liegt für ein Feature genau dann vor, wenn keine zulässige Konfiguration existiert, in der das Feature ausgewählt ist.
- Eine Anomalie vom Typ *falsch-optional* (engl. false optional) liegt bei einem Feature genau dann vor, wenn es zwar als optional spezifiziert wurde, jedoch in jeder Konfiguration ausgewählt ist.
- Eine Anomalie vom Typ *Kern-Feature* (engl. core feature) liegt für ein Feature genau dann vor, wenn das Feature in allen zulässigen Konfigurationen ausgewählt ist.
- Eine Anomalie vom Typ *totes Kardinalitätsintervall* (engl. dead cardinality interval) liegt für ein Feature-Instanz-Kardinalitätsintervall in einem kardinalitätsbasierten Feature-Modell genau dann vor, wenn für ein betrachtetes Teilintervall keine zulässige Konfiguration gefunden werden kann.

Folgende Beispiele demonstrieren die beschriebenen Anomalietypen.

Beispiel 2.22 (Anomalien in Feature-Modellen)
Anomalie vom Typ totes Feature: In keiner Konfiguration des Feature-Modells in Abbildung 2.3 ist das Feature `Offline` ausgewählt. Daher weist das Feature-Modell eine Anomalie vom Typ *totes Feature* auf. Die Anomalie kann darauf hindeuten, dass (i) das Feature `Offline` für die Problemdomäne keine Relevanz hat und somit entfernt werden kann oder (ii) ein Spezifikationsfehler vorliegt, der dazu führt, dass das Feature `Offline` nicht wählbar ist.
Anomalie vom Typ falsch-optionales Feature und Kern-Feature: In dem Feature-Modell in Abbildung 2.3 ist das Feature `Interface` als optional spezifiziert, obwohl es in jeder Konfiguration ausgewählt sein muss. Das Feature `Interface` ist somit ein Kern-Feature und weist eine Anomalie vom Typ *falsch-optionales Feature* auf. Eine Ursache für diese Anomalie kann eine fehlerhaft spezifizierte Dekompositionsbeziehung sein.
Anomalie vom Typ totes Kardinalitätsintervall: In dem Feature-Modell in Abbildung 2.3 müssen die Feature-Typen `Node`, `DisseminationStrategy` und `Channels` in einer Konfiguration jeweils mindestens einmal instanziiert sein. In Summe sind

somit immer mindestens drei Feature-Typen in einer Konfiguration präsent. Das Gruppentyp-Kardinalitätsintervall von Feature `DisseminationGroup` weist somit eine Anomalie vom Typ *totes Kardinalitätsintervall* für das Teilintervall $[0, 2]$ auf.

Die bestehende Literatur enthält vor allem Analyseverfahren zur Erkennung von Anomalietypen in herkömmlichen (nicht-erweiterten) Feature-Modellen. Die Analyse von erweiterten kardinalitätsbasierten Feature-Modellen mit komplexen Constraints über numerischen Features und Feature-Instanzen ist besonders herausfordernd, da (i) der Konfigurationsraum durch unbeschränkte Wertebereiche numerischer Features oder unbeschränkte Kardinalitätsintervalle beliebig groß werden kann (siehe Herausforderungen 1 und 4), (ii) die Kombination von verschiedenen Kardinalitätsintervallen zu neuartigen, bisher nicht betrachteten, Anomalietypen führt (siehe Herausforderung 2) und (iii) keine effizienten Verfahren zur Behandlung von ganzzahligen und reellwertigen Attributen existieren (siehe Herausforderung 3).

2.3 DYNAMISCHE SOFTWARE-PRODUKTLINIEN

Dynamische Software-Produktlinien (DSPLs) sind ein systematischer und ganzheitlicher Entwicklungsansatz zur Konzeption selbst-adaptiver Systeme [23, 64, 65]. Der Entwicklungsprozess von SPLs, der in Abschnitt 2.2.2 beschrieben wurde, wird durch DSPLs erweitert und auf adaptive sowie selbst-adaptive Systeme übertragen [39]. Die Kernerweiterungen von DSPLs gegenüber herkömmlichen SPLs sind (i) dynamische Bindungszeiten von Konfigurationsoptionen, (ii) Konzepte zur expliziten Charakterisierung von Kontextabhängigkeiten und Kontextvariabilität sowie (iii) Möglichkeiten zur Spezifikation von Laufzeitvariabilität [36]. Im Folgenden beschreiben wir diese Kernerweiterungen genauer.

2.3.1 *Dynamische Bindungszeit*

Als Bindungszeit wird derjenige Zeitpunkt bezeichnet, in dem das System an eine bestimmte Produktvariante gebunden wird [140]. In DSPLs werden Varianten dynamisch zur Laufzeit gewechselt, Konfigurationsoptionen werden zur Laufzeit konfiguriert und dynamisch an Komponenten gebunden. Es müssen somit Methoden zur Spezifikation und Durchsetzung dynamischer Bindungszeiten unterstützt werden. Es wird zwischen *statischen* und *dynamischen Features* unterschieden [90]. Ein statisches Feature kann nur zur Entwurfszeit konfiguriert werden. Dynamische Features können hingegen zur Laufzeit an- oder abgewählt werden [44]. Darüber hinaus wurden Konzepte vorgeschlagen, um mehrere Bindungszeiten und Abhängigkeiten zwischen Bindungszeiten zu spezifizieren [29, 46, 126].

Im Rahmen dieser Arbeit liegt der Fokus auf der Validierung und Analyse struktureller Abhängigkeiten zwischen Elementen integrierter Produktlinien-Spezifikationen. Wir

nehmen daher an, dass Konfigurationsoptionen dynamisch gebunden werden können und nur ein Bindungszeitpunkt existiert.

2.3.2 *Explizite Charakterisierung von Kontextabhängigkeiten und Kontextvariabilität*

DSPLs stellen Konzepte zur expliziten Charakterisierung von Kontextvariabilität bereit. Der Begriff Kontext fasst alle Informationen zusammen, die geeignet sind, um die Situation der Umgebung eines Systems zu charakterisieren und potentiell das Systemverhalten zu beeinflussen [1]. Ein Kontextvariabilitätsmodell erfasst die Gemeinsamkeiten und Unterschiede des Kontexts eines Systems und spezifiziert Abhängigkeiten zwischen System- und Kontextcharakteristika [66]. In der Literatur werden einige Ansätzen vorgeschlagen, um Variabilität des Kontexts sowie Abhängigkeiten zu Systemcharakteristika zu spezifizieren [104]: In [2, 55] werden Kontextinformationen als separates Feature-Modell beschrieben und Abhängigkeiten zwischen Feature-Modellen werden mithilfe von Event-Condition-Regeln spezifiziert. In [100] werden Kontextabhängigkeiten durch zusätzliche Constraints codiert. In [66, 130] werden Kontext-Feature-Modelle vorgeschlagen, in denen Teile des Feature-Modells die Variabilität des Kontexts erfassen und Abhängigkeiten zu (rekonfigurierbaren) Systemcharakteristika spezifizieren. Wir verwenden im Rahmen dieser Arbeit Kontext-Feature-Modelle, da sie eine einheitliche Repräsentation von System- und Kontextvariabilität ermöglichen.

Abbildung 2.6 zeigt das Kontext-Feature-Modell des in dieser Arbeit betrachteten Beispiels, anhand dessen wir im Folgenden Syntax und Semantik erläutern. In Kontext-Feature-Modellen wird zwischen *System-* und *Kontext-Features* unterschieden [131], wobei System-Features unterhalb des speziellen Features `System` und Kontext-Features unterhalb des speziellen Features `Context` angeordnet sind. In erweiterten Feature-Modellen können zusätzlich (zu numerische System-Features) numerische Kontext-Features verwendet werden, um numerische Kontextvariabilität zu charakterisieren. Die *Kontextkonfiguration* bezeichnet die partielle Konfiguration bestehend aus allen Features unterhalb des speziellen Features `Context`. Entsprechend bezeichnet die *Systemkonfiguration* die partielle Konfiguration bestehend aus allen Features unterhalb des speziellen Features `System`.

System-Features sind durch die Adaptionslogik rekonfigurierbar, Kontext-Features werden hingegen anhand der gegenwärtigen Systemumgebung fortwährend angepasst. Mithilfe der in Abschnitt 2.2.3 gezeigten Syntax lassen sich Abhängigkeiten zwischen System- und Kontext-Features durch Cross-Tree-Constraints spezifizieren. Falls eine Änderung der Kontextkonfiguration zu einer inkonsistenten Gesamtkonfiguration führt, muss die Systemkonfiguration angepasst werden. Die Identifikation einer konsistenten Gesamtkonfiguration für eine durch die Systemumgebung gegebene Kontextkonfiguration stellt ein wesentliches Planungsproblem in selbst-adaptiven Systemen dar [71].

Beispiel 2.23 (Kontext-Feature-Modelle) Abbildung 2.6 zeigt das Kontext-Feature-Modell des durchgängigen Beispiels. Zusätzlich zu der Spezifikation, die in Abbildung 2.4 gezeigt ist, wurden die speziellen Features `System` und `Context` eingeführt, die System- bzw. Kontext-Features gruppieren. Cross-Tree-Constraints zwischen Kontext- und System-Features charakterisieren Kontextabhängigkeiten des Systems. So darf beispielsweise, falls das Kontext-Feature `Emergency` gewählt ist, nicht das System-Feature `ProbabilisticBroadcast` gewählt sein. Weiterhin muss, falls das Kontext-Feature `Normal` gewählt ist, das System-Feature `Reliable` und `ProbabilisticBroadcast` gewählt werden.

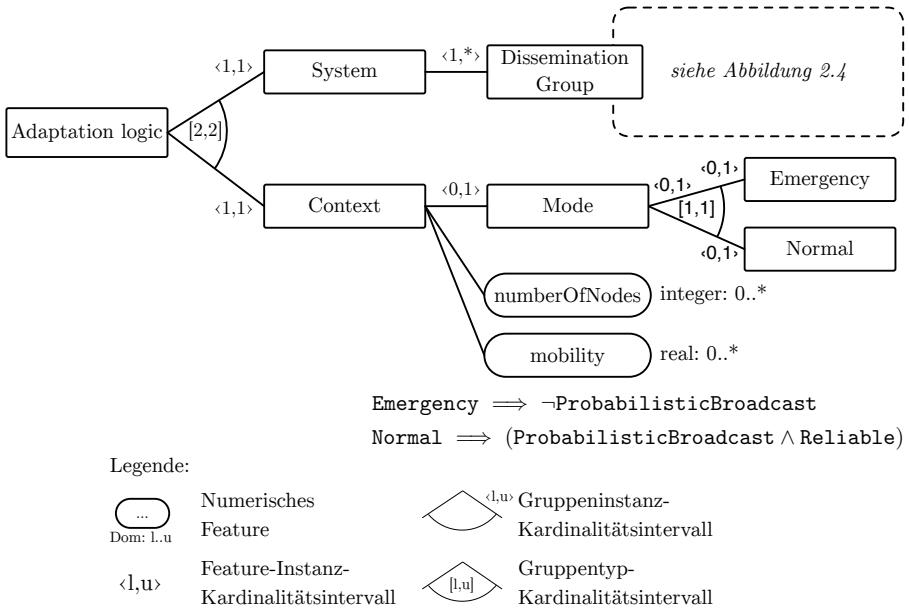


Abbildung 2.6: Kontext-Feature-Modell des betrachteten Beispiels

2.3.3 *Spezifikation der Laufzeitvariabilität*

In DSPLs werden Variabilitätsmodelle zur Laufzeit verwendet, um Adaptionsentscheidungen zu identifizieren. Das System muss hierfür eine geeignete Architektur zur Verfügung stellen, die explizite Unterstützung anbietet, um Rekonfigurationen zur Laufzeit umzusetzen und Konfigurationsoptionen an dynamisch abgeleitete Produkt-Varianten zu binden [71]. Bei selbst-adaptiven Systemen hat sich hierzu die in Abschnitt 2.1 beschriebene MAPE-Schleife etabliert, die in Verbindung mit DSPLs eine breite Verwendung zur Strukturierung der modellbasierten Adaptionslogik findet [22]. Im Folgenden beschrei-

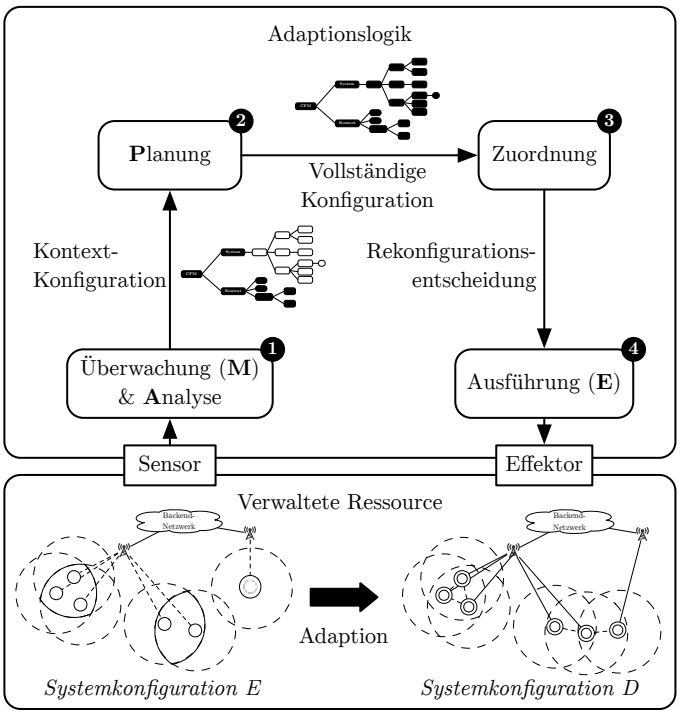


Abbildung 2.7: MAPE-basierte Rekonfiguration in DSPLs

ben wir anhand von Abbildung 2.7 die Phasen sowie Modellierungsartefakte innerhalb der MAPE-Schleife einer DSPL.

- Überwachung & Analyse (engl. monitoring & analysis): In der Überwachungsphase sammeln Sensoren uninterpretierte Kontextdaten der verwalteten Ressourcen, die anschließend während der Analysephase in eine Kontextkonfiguration übersetzt werden (siehe ❶ in Abbildung 2.7). Die Kontextkonfiguration wird im nächsten Schritt an die Planungskomponente übergeben.

- Planung (engl. planning): Während der Planungsphase wird ausgehend von der zuvor identifizierten Kontextkonfiguration eine konsistente Systemkonfiguration ermittelt (siehe ❷ in Abbildung 2.7). Hierbei wird auf die Spezifikation des Kontext-Feature-Modells zurückgegriffen, das auf die vorgegebene Kontextkonfiguration fixiert wird. Zusätzlich können auch nicht-funktionale Charakteristiken berücksichtigt werden, um konsistente Systemkonfigurationen zu priorisieren [50, 146]. Das Ergebnis der Planungsphase ist eine vollständige Konfiguration des Kontext-Feature-Modells (Systemkonfiguration und Kontextkonfiguration), die einen konsistenten Zielzustand des Systems repräsentiert.

  Die Planung mittels Kontext-Feature-Modellen zur Laufzeit setzt voraus, dass zu jeder Kontext-Konfiguration mindestens eine konsistente Systemkonfiguration

existiert. Die Identifikation von Kontextkonfigurationen zur Entwurfszeit, für die keine Systemkonfiguration existiert, ist somit ein wichtiges Analyseziel, das wir in dieser Arbeit behandeln (siehe Herausforderungen 1 und 2).

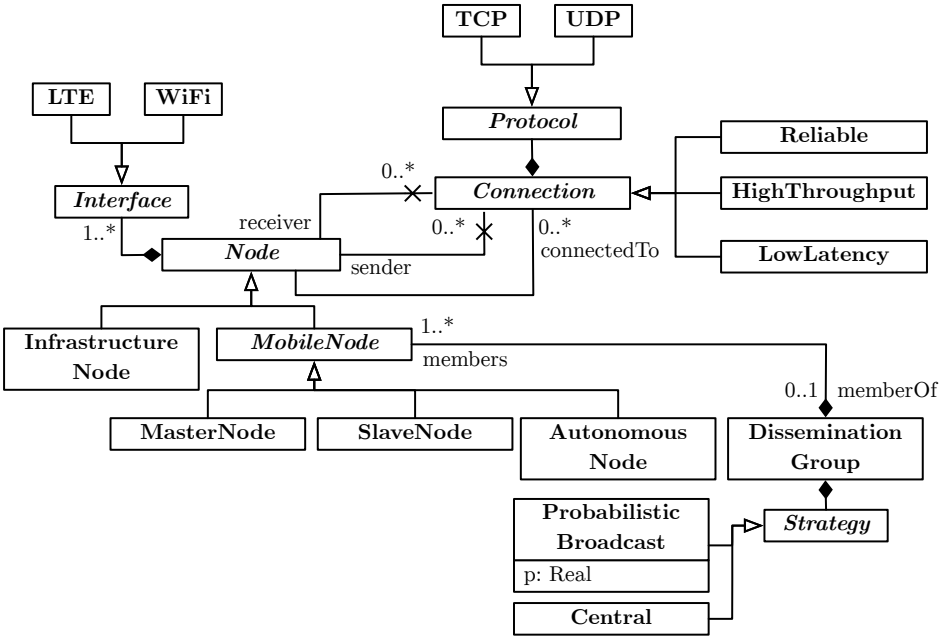
- Zuordnung und Ausführung (engl. execution): Die Konfiguration der Planungsphase wird im Vorfeld der Ausführungsphase einer oder mehrerer Komponenten der verwalteten Ressource samt Parameterbelegung zugeordnet (siehe ❸ in Abbildung 2.7). Hierfür werden die Spezifikation der Lösungsraumarchitektur und das Zuordnungsmodell verwendet, das Konfigurationsoptionen des Problemraums den Komponenten des Lösungsraums zuordnet.

  Während der Ausführungsphase wird schließlich eine zulässige Instanz der Lösungsraumreferenzarchitektur ermittelt und an Effektoren der verwalteten Ressource übergeben, welche die Rekonfigurationsentscheidung umsetzen (siehe ❹ in Abbildung 2.7). Dies setzt voraus, dass für eine gegebene Systemkonfiguration des Problemraums mindestens eine zulässige Instanz der Lösungsraumspezifikation existiert. Die Identifikation derartiger Inkonsistenzen ist ein wichtiges Analyseziel, das wir im Rahmen dieser Arbeit ebenfalls behandeln.

2.4 SPEZIFIKATION DES LÖSUNGSRAUMS

Die ganzheitliche Spezifikation der Konsistenzeigenschaften eines selbst-adaptiven Systems umfasst neben der zuvor vorgestellten Problemraumspezifikation, welche System- und Kontextvariabilität erfasst, auch die Lösungsraumspezifikation, welche die Architektur der zu adaptierenden verwalteten Ressourcen beschreibt. Der Lösungsraum (siehe Abschnitt 2.2.2) umfasst Implementierungsartefakte, Dokumentationen sowie Architektur- und Verhaltensmodelle. Der Fokus dieser Arbeit liegt auf der Analyse von Architekturen sowie struktureller Eigenschaften selbst-adaptiver Systeme. Verhaltensmodelle werden daher im Folgenden nicht explizit berücksichtigt. Klassendiagramme, die Teil des weitverbreiteten Spezifikationsstandards Unified Modeling Language (UML) [128] sind, eignen sich, um strukturelle Aspekte von Software-Architekturen zu spezifizieren [88].

Beispiel 2.24 (UML-Klassendiagramm zur Spezifikation von Software-Komponenten des Lösungsraums) Abbildung 2.8 zeigt das UML-Klassendiagramm der Architektur des betrachteten selbst-adaptiven Kommunikationssystems. Klassen repräsentieren hierbei Software-Komponenten des Lösungsraums, deren Eigenschaften durch Attribute charakterisiert werden. Die Klasse `ProbabilisticBroadcast` repräsentiert die Funktionalität eines probabilistischen Broadcast-Protokolls. Der Parameter `p` gibt die Wahrscheinlichkeit an, mit der Nachrichten verworfen werden. Durch Anpassen des Parameters kann somit das Verhalten der Software-Komponente beeinflusst werden. Der Parameter ist als Attribut `p` der Klasse `ProbabilisticBroadcast` im Klassendiagramm spezifiziert.



❶ **context** SlaveNode **inv**: **not** self.connectedTo->exists(c |
c.receiver.**oclIsTypeOf(**AutonomousNode**)** **or** c.sender.**oclIsTypeOf(**AutonomousNode**)**
or c.receiver.**oclIsTypeOf(**InfrastructureNode**)** **or** c.sender.**oclIsTypeOf(**InfrastructureNode**)**)

❷ **context** MasterNode **inv**: **self**.connectedTo->**one**(c |
c.sender.**oclIsTypeOf(**InfrastructureNode**)** **or** c.receiver.**oclIsTypeOf(**InfrastructureNode**)**)

❸ **context** ProbabilisticBroadcast **inv**: 0 < **self**.p **and** **self**.p < 1

❹ **context** Connection **inv**: Connection.**allInstances()**->**forAll**(c1, c2: Connection |
c1 <> c2 **implies** **not**(c1.sender = c2.sender **and** c1.receiver = c2.receiver))

❺ **context** Node **inv: self.**connectedTo **-> forAll(c | c.**receiver = **self xor c.**sender = **self)**

❻ **context** Connection **inv**: **self**.receiver <> **self**.sender

❼ **context** DisseminationGroup **inv**: **self**.members ->**one**(n | n.**oclIsTypeOf**(MasterNode))

❽ **context** DisseminationGroup **inv**: **not** **self**.members ->**exists**(n | n.**oclIsTypeOf**(AutonomousNode))

Abbildung 2.8: UML-Klassendiagramm des betrachteten Beispiels

Weiterhin können in Klassendiagrammen (i) Assoziationsbeziehungen, (ii) Kompositionsbeziehungen und (iii) Generalisierungsbeziehungen zwischen Klassen spezifiziert werden. Im Folgenden beschreiben wir die Syntax sowie Semantik der genannten Beziehungstypen anhand von Beispielen.

- Eine *Assoziationsbeziehung* zwischen zwei Klassen repräsentiert eine Kopplung zweier Komponenten (z. B. durch Kommunikationsverbindungen). In der graphischen Syntax werden Assoziationsbeziehungen durch Kanten zwischen Klassen spezifiziert. Die Enden einer Assoziationskante können mit Multiplizitätsintervallen der Form $l..u$ annotiert werden (falls kein Multiplizitätsintervall angegeben ist, wird 1..1 als Intervall angenommen). Multiplizitätsintervalle geben an, mit wie vielen Instanzen der assoziierten Klasse eine Instanz der assoziierenden Klasse mindestens (Untergrenze l) oder höchstens (Obergrenze u) in Beziehung steht. Falls als Obergrenze eines Multiplizitätsintervalls das Symbol $*$ verwendet wird, können beliebig viele Instanzen assoziiert werden. Zusätzlich kann eine Assoziationsbeziehung mit einem Rollennamen beschriftet sein.

- *Kompositionsbeziehungen* zwischen Klassen repräsentieren ist-Teil-von-Beziehungen zwischen einer Teil-Komponente und einer Aggregat-Komponente des Lösungsraums. In der graphischen Syntax wird eine Kompositionsbeziehungen durch eine Kante repräsentiert, wobei das Kantenende der Aggregat-Komponente durch eine ausgefüllte Raute gekennzeichnet ist. Eine Teil-Komponente kann zu höchstens einer Aggregat-Komponente in Beziehung stehen. Das Multiplizitätsintervall der Aggregat-Komponente darf somit nur 0..1 oder 1..1 sein. Eine Aggregat-Komponente kann hingegen potentiell über beliebig viele Teil-Komponenten verfügen. Für das Multiplizitätsintervall der Teil-Komponente einer Kompositionsbeziehung besteht daher keine Einschränkung.

- *Generalisierungsbeziehungen* zwischen Klassen repräsentieren Vererbungsbeziehungen zwischen einer allgemeinen und einer speziellen Komponente. Instanzen der speziellen Klasse (die Instanzen der speziellen Komponente repräsentieren) sind immer auch Instanzen der allgemeinen Klasse. Die spezielle Klasse erbt somit alle Merkmale der allgemeinen Klasse. Falls eine Klasse als abstrakt gekennzeichnet ist, kann diese Klasse nicht instanziiert werden. Abstrakte Klassen werden durch kursive Klassenbezeichner in der graphischen Syntax gekennzeichnet.

Beispiel 2.25 (Beziehungen in UML-Klassendiagrammen)
Assoziationsbeziehung: In Abbildung 2.8 sind die Klassen `Node` und `Connection` durch eine Assoziationsbeziehung mit dem Namen `connectedTo` verbunden. Die Enden der Assoziationskante sind mit den Multiplizitäten 1..1 (implizit) und 1..$*$ annotiert. Instanzen der Klasse `Node` müssen somit mit jeweils mindestens einer

Instanz der Klasse `Connection` in Beziehung stehen (Untergrenze). Die maximale Anzahl der Assoziationsbeziehungen eines Objekts der Klasse `Node` zu Objekten der Klasse `Connection` ist hingegen nicht beschränkt (Obergrenze). Entsprechend müssen Instanzen der Klasse `Connection` jeweils mit genau einer Instanz der Klasse `Node` über die Assoziationsbeziehung `connectedTo` in Beziehung stehen. Zusätzlich verfügt die Klasse `Connection` über zwei Assoziationsbeziehungen zur Klasse `Node`, die den Sende- (`sender`) und Empfangsknoten (`receiver`) einer Kommunikationsverbindung repräsentieren.

Kompositionsbeziehung: In Abbildung 2.8 steht die Klasse `DisseminationGroup` in einer Kompositionsbeziehung zu der Klasse `MobileNode`. Die Klasse `DisseminationGroup` repräsentiert eine Aggregat-Komponente, die aus Teil-Komponenten besteht, die durch die Klasse `MobileNode` repräsentiert werden. Durch das Multiplizitätsintervall 0..1 der Aggregat-Komponente ist sichergestellt, dass eine Instanz der Klasse `MobileNode` höchstens einer Instanz der Klasse `DisseminationGroup` zugeordnet ist. Demgegenüber müssen aufgrund des Multiplizitätsintervalls 1..$*$ der Teil-Komponente, Instanzen der Klasse `DisseminationGroup` mindestens eine Instanz der Klasse `MobileNode` referenzieren. Die maximale Anzahl der referenzierten Instanzen der Klasse `MobileNode` ist nicht eingeschränkt.

Generalisierungsbeziehung: In Abbildung 2.8 besteht zwischen der Klasse `MobileNode` und den Klassen `MasterNode`, `SlaveNode` und `AutonomousNode` jeweils eine Generalisierungsbeziehung. Die Klasse `MobileNode` repräsentiert hierbei jeweils die allgemeine Komponente. Die Klasse `MobileNode` steht wiederum in einer Generalisierungsbeziehung zu der Klasse `Node`. Instanzen der Klassen `MasterNode`, `SlaveNode` und `AutonomousNode` sind somit Instanzen der Klassen `MobileNode` und `Node`. Instanzen der speziellen Klasse verfügen über die gleichen Attribute und Assoziationsbeziehungen wie die Klassen `MobileNode` und `Node`.

Zwischen Instanzen von Klassen können Einschränkungen existieren, die sich nicht mit den zuvor vorgestellten Sprachmitteln von UML-Klassendiagrammen charakterisieren lassen. Zur Spezifikation komplexer Einschränkungen zwischen Instanzen von Klassen hat sich die Sprache Object Constraint Language (OCL) etabliert [33]. Ein OCL-Ausdruck besteht aus einem Anwendungskontext, der die Anwendungsbedingung bestimmt, und einer Operation, die Eigenschaften von Instanzen definiert oder modifiziert. Im Folgenden erläutern wir anhand der Abbildung 2.8 exemplarisch die Semantik von OCL-Ausdrücken. Für eine vollständigen Überblick über OCL sei an dieser Stelle auf [33] verwiesen.

Beispiel 2.26 (OCL-Ausdrücke zur Spezifikation komplexer Einschränkungen zwischen Instanzen von Klassen) In Abbildung 2.8 werden zusätzliche Einschränkungen zwischen Instanzen von Klassen durch folgende OCL-Ausdrücke spezifiziert:

- Knoten, die als Slave in einer Disseminationsgruppe fungieren, sollen keine Verbindungen zu Infrastrukturknoten oder autonomen Knoten aufbauen können. Der OCL-Ausdruck ❶ beschreibt diese Anforderung. Durch den Ausdruck wird verhindert, dass Assoziationsbeziehungen zwischen Instanzen der Klasse `SlaveNode` und Instanzen der Klassen `InfrastructureNode` und `AutonomousNode` existieren können.
- Durch den OCL-Ausdruck ❷ wird sichergestellt, dass jeder Knoten, der in einer Disseminationsgruppe als Master fungiert, mit genau einem Infrastrukturknoten verbunden ist.
- Die Sprache OCL erlaubt darüber hinaus die Einschränkung der Wertebereiche von Attributen. Der OCL-Ausdruck ❸ schränkt beispielsweise das Attribut `p` der Klasse wie folgt ein: $0 \leq p \leq 1$.
- Durch den OCL-Ausdruck ❹ wird sichergestellt, dass Kommunikationsverbindungen zwischen beliebigen Knoten nicht mehrfach auftreten können.
- Durch die OCL-Ausdrücke ❺ und ❻ wird sichergestellt, dass Knoten keine Kommunikationsverbindungen zu sich selbst aufbauen können.
- Die OCL-Ausdrücke ❼ und ❽ schränken die Kompositionsbeziehung einer Disseminationsgruppe zu Instanzen der Klasse `MobileNode` so ein, dass je Disseminationsgruppe genau ein Knoten als Master fungiert und keine Instanz der Klasse `AutonomousNode` auftreten kann.

Im betrachteten Beispiel entsprechen Instanzen der Klassen konkreten Komponenteninstanzen. Eine Instanz des Klassendiagramms ist konsistent, wenn alle geforderten strukturellen Eigenschaften und alle OCL-Ausdrücke erfüllt sind. Zur Repräsentation einer Instanz eines Klassendiagramms wurden im UML-Standard Objektdiagramme eingeführt [128]. Ein Objektdiagramm besteht aus Objekten, die Instanzen von Klassen repräsentieren, und Beziehungen zwischen Objekten, die Instanzen von Assoziationen oder Kompositionsbeziehungen sind. Objekte dürfen nur zu konkreten Klassen existieren (beispielsweise darf keine Instanz der abstrakten Klasse `Node` existieren). Die Menge von Objektdiagrammen, die konsistent zu einem Klassendiagramm sind, ist potentiell unbeschränkt groß.

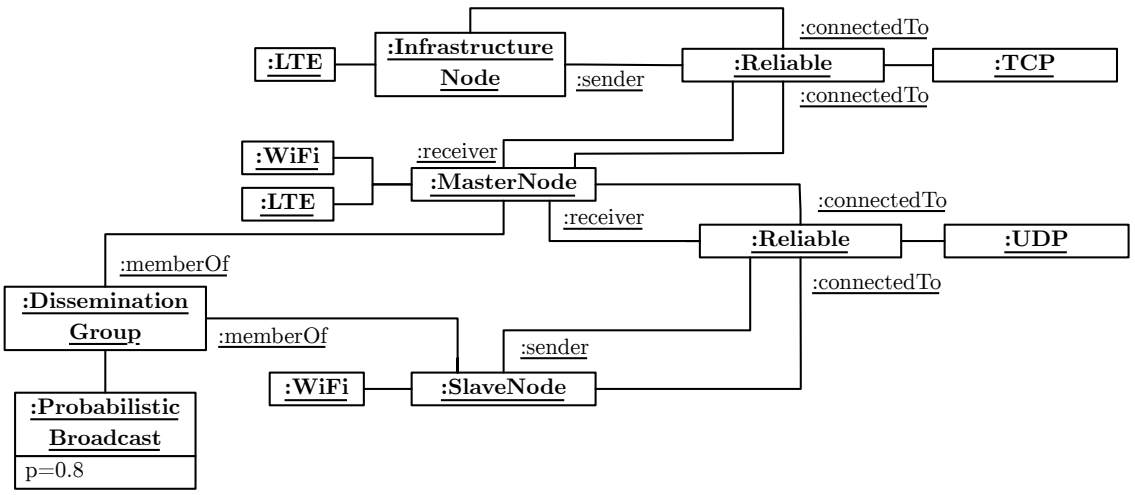


Abbildung 2.9: UML-Objektdiagramm des betrachteten Beispiels

Beispiel 2.27 (Objektdiagramm eines UML-Klassendiagramms) Abbildung 2.9 zeigt ein mögliches Objektdiagramm des Klassendiagramms, das in Abbildung 2.8 dargestellt ist. Objekte stehen entsprechend der Spezifikation im Klassendiagramm zueinander in Beziehung und Attribute sind mit Werten belegt.

Insbesondere existieren im Objektdiagramm Objekte, die Instanzen der Klassen `ProbabilisticBroadcast` und `Reliable` repräsentieren. Die Klassen repräsentieren hierbei Komponenten, die den gleichnamigen Features im Problemraum zugeordnet sind. Da das Feature `ProbabilisticBroadcast` und das Feature `Reliable` im Feature-Diagramm, das in Abbildung 2.4 gezeigt ist, aufgrund einer Exclude-Beziehungen nicht gleichzeitig in einer Konfiguration auftreten dürfen, repräsentiert das Objektdiagramm keine konsistente Variante der DSPL. Das Beispiel zeigt somit, dass eine separate Problem- und Lösungsraumbetrachtung zur Validierung und Analyse struktureller Eigenschaften nicht ausreicht.

Die in diesem Abschnitt vorgestellte Lösungsraumspezifikation des motivierenden Beispiels ist geeignet, die Referenzarchitektur aller Varianten der DSPL zu charakterisieren. Das Beispiel 2.27 hat jedoch gezeigt, dass Instanzen der Lösungsraumspezifikation (repräsentiert als Objektdiagramm) nicht zwangsläufig einer konsistenten Variante der DSPL entsprechen. Für die Validierung und Analyse ist es daher unabdingbar, dass Problem- und Lösungsraumspezifikationen integriert betrachtet werden (siehe Herausforderung 1).

3

INTEGRIERTE PRODUKTLINIEN-SPEZIFIKATION IN CLAFER

In diesem Kapitel beschreiben wir ausgehend von der integrierten Produktlinien-Spezifikation des betrachteten selbst-adaptiven Kommunikationssystems die Elemente der Sprache Clafer. Im zweiten Schritt charakterisieren wir ausgehend von Anomalietypen, die in Spezifikationen des Problemraums auftreten, neuartige Anomalietypen in Clafer-Spezifikationen.

3.1 SPRACHELEMENTE VON CLAFER

Die Validierung struktureller Konsistenzeigenschaften und die Erkennung potentieller Anomalien erfordert eine geeignete Repräsentation des Problem- und Lösungsraums. Zur Repräsentation des Lösungsraums wurden in Abschnitt 2.2 und Abschnitt 2.3 Variabilitätsmodelle eingeführt. Darüber hinaus wurden zur Repräsentation des Lösungsraums in Abschnitt 2.4 UML-Klassendiagramme präsentiert. Beispiel 2.27 hat jedoch demonstriert, dass es nicht ausreicht, Problem- und Lösungsraumspezifikationen getrennt zu analysieren, da zahlreiche Inkonsistenzen erst bei einer integrierten Betrachtung erkannt werden können. Die explizite Spezifikation von Abhängigkeiten zwischen Problem- und Lösungsraumelementen ist daher eine wesentliche Voraussetzung, um eine integrierte Validierung und Analyse zu ermöglichen. Sie ist jedoch aufgrund folgender Faktoren herausfordernd (siehe Herausforderung 1):

(i) Sowohl Konfigurationsoptionen des Problemraums als auch Lösungsraumkomponenten sind potentiell mehrfach instanziierbar. Darüber hinaus können Abhängigkeiten zwischen Instanzen existieren. Beispielsweise kann eine Komponenteninstanz, die einen Kommunikationsknoten repräsentiert, individuell konfiguriert werden und Abhängigkeiten zu anderen Komponenteninstanzen aufweisen.

(ii) Im Lösungsraum bestehen (rekursive) orthogonale Vererbungs- und Kompositionsbeziehungen zwischen Komponenteninstanzen. Beispielsweise besteht eine Komponente aus Teil-Komponenten, die Eigenschaften von einer gemeinsamen Komponente ableiten.

(iii) Es bestehen komplexe Abhängigkeiten zwischen (multiinstanziierbaren) numerischen Features und Attributen von Komponenten. Beispielsweise sind Sensordaten des Lösungsraums numerischen Kontext-Features im Lösungsraum zugeordnet, die Auswirkungen auf rekonfigurierbare System-Features haben. System-Features sind wiederum Komponenten des Lösungsraums zugeordnet.

Abhängigkeiten zwischen Problem- und Lösungsraum können als Variabilitätsannotationen an Lösungsraumartefakten [46, 68, 139, 160] in Form eigenständiger Zuordnungsmodelle [25, 67] oder in einer integrierten Repräsentation [9, 10, 61, 137] beschrieben werden. Eine integrierte Repräsentation ermöglicht es, Problem- und Lösungsraumartefakte sowie Abhängigkeiten in einer einheitlichen Spezifikationssprache zu charakterisieren und anschließend zu validieren. Diese Art integrierter Repräsentation von Problem- und Lösungsraum wird im Folgenden als *integrierte Software-Produktlinien-Spezifikation* bezeichnet. Die Sprache Clafer stellt einen Vertreter integrierter Software-Produktlinien-Spezifikationen dar.

Clafer (Class, feature, reference) kombiniert Sprachelemente aus UML-Klassendiagrammen und erweiterten kardinalitätsbasierten Feature-Modellen in einer einheitlichen Repräsentation [10]. Constraints über Elemente der Spezifikation können in Prädikatenlogik erster Stufe spezifiziert werden. Clafer eignet sich somit zur integrierten Spezifikation von Problem- und Lösungsraumartefakten sowie deren Zuordnung. Im Folgenden beschreiben wir die Sprachelemente sowie die Semantik von Clafer anhand der in Listing 3.1 gezeigten Spezifikation des betrachteten selbst-adaptiven Kommunikationssystems. Für eine Beschreibung des vollständigen Sprachumfangs sei an dieser Stelle auf [9] verwiesen.

Eine Clafer-Spezifikation besteht aus einer Menge von sogenannten Clafer-Definitionen, die zueinander in Beziehung stehen. Eine Clafer-Definition kann entweder eine Konfigurationsoption aus dem Problemraum oder eine Komponente aus dem Lösungsraum repräsentieren. Diese Unterscheidung wird jedoch nicht explizit durch die Sprache Clafer abgebildet. Instanzen von Clafer-Definitionen können zu anderen Instanzen über (i) Kompositionsbeziehungen, (ii) Referenzen, (iii) Vererbungsbeziehungen und (iv) Constraints in Beziehung gesetzt werden. Im Folgenden beschreiben wir die genannten Sprachelemente.

Kompositionsbeziehungen

Kompositionsbeziehungen zwischen Clafer-Definitionen werden in Clafer-Spezifikationen syntaktisch durch Einrückungen repräsentiert, die Kind-Clafer-Definitionen in ihren Eltern-Clafer-Definition schachteln. In der Sprache Clafer entsprechen Kompositionsbeziehungen den Dekompositionsbeziehungen zwischen Features in kardinalitätsbasierten Feature-Modellen und Kompositionsbeziehungen zwischen Klassen aus UML-Klassendiagrammen. Eine Clafer-Definition verfügt in der Syntax auf der rechten Seite über ein Multiplizitätsintervall $l..u$, das die minimale und maximale Anzahl von erlaubten Instanzen der Kind-Clafer-Definition pro Instanz der Eltern-Clafer-Definition angibt.

```
abstract SystemElement (1)

abstract Node : SystemElement 0..* (2)
  connections -> Connection 0..* (3)
    [ !(#(this.dref) >= 3 && #(this.dref) <= 4) ] (4)
    [ this.parent = this.dref.receiver.dref xor this.parent = this.dref.sender.dref ] (5)
  or Interface 1..1 (6)
    LTE 0..1
    Wifi 0..1

abstract MobileNode : Node 0..* (7)
  memberOf -> DisseminationGroup 0..1
    [ this.parent in this.dref.members.dref ] (8)

SlaveNode : MobileNode 0..* (9)
  [ some this.memberOf ]
  [ no this.Interface.LTE && some this.Interface.Wifi ]
  [ MasterNode in this.connections.dref.receiver.dref ]

AutonomousNode : MobileNode 0..* (10)
  [ some this.Interface.LTE] (11)
  [ one InfrastructureNode **
      (this.connections.dref.receiver.dref ++ this.connections.dref.sender.dref) ]  (12)

MasterNode : MobileNode 0..* (13)
  [ some this.memberOf ]
  [ some this.Interface.Wifi && some this.Interface.LTE ]
  [ one InfrastructureNode **
      (this.connections.dref.receiver.dref ++ this.connections.dref.sender.dref) ]

InfrastructureNode : Node 0..* (14)
  [ some this.Interface.LTE && no this.Interface.Wifi ]
  [ no this.connections.dref.sender.dref ** SlaveNode ] (15)
```

Listing 3.1: Clafer-Spezifikation des motivierenden Beispiels

```
abstract Connection : SystemElement 0..*
  sender -> Node 1..1
  receiver -> Node 1..1
  [ this in this.sender.dref.connections.dref && this in this.receiver.dref.connections.dref ] 16
  nodes -> Node 0..2
    [ this.parent.sender.dref ++ this.parent.receiver.dref in this.dref ] 17
  xor Protocol 1..1
    TCP 0..1
    UDP 0..1

[ no disj a; b: Connection | a.receiver.dref = b.receiver.dref && a.sender.dref = b.sender.dref ] 18

Reliable : Connection 0..*
  [ no DisseminationGroup.Strategy.ProbabilisticBroadcast ]

HighThroughput : Connection 0..*

LowLatency : Connection 0..*

DisseminationGroup : SystemElement 0..*
  members -> MobileNode -- AutonomousNode 1..* 19
    [ this.dref.memberOf.dref = this.parent ]
  [ one this.members.dref ** MasterNode ] 20
  [ 2 <= #(this.members.dref.connections) && #(this.members.dref.connections) <= 30 ] 21
  [ 1 <= #(this.members.dref.connections.dref ** Reliable) &&
      #(this.members.dref.connections.dref ** Reliable) <= 8 ]
  xor Strategy 1..1
    Central 0..1
    ProbabilisticBroadcast 0..1
      p -> real 1..1 22
      [ 0 <= this.p.dref && this.p.dref <= 1 && this.p.dref != 0.5 ] 23
      [ no Reliable ]

0..* Context 1..1 24
  numberOfNodes -> integer 1..1 25
    [this = #Node ] 26
  mobility -> real 0..1 27
  xor Mode 0..1 28
    Emergency 0..1
      [ no ProbabilisticBroadcast ] 29
    Normal 0..1 30
      [ some ProbabilisticBroadcast && some Reliable ] 31
  [ some this.Mode <=> some this.mobility ] 32
```

Listing 3.1 (fortgesetzt)**:** Clafer-Spezifikation des motivierenden Beispiels

In der Sprache Clafer entsprechen Multiplizitätsintervalle den Feature-Instanz-Kardinalitätsintervallen aus kardinalitätsbasierten Feature-Modellen [45]. Der Ausdruck $*$ stellt eine vereinfachte Schreibweise für das Multiplizitätsintervall $0..*$ dar. Ein Gruppenkardinalitätsintervall $l..u$, das auf der linken Seite einer Clafer-Definition annotiert ist, schränkt die minimale und maximale Anzahl der Instanzen der geschachtelten Clafer-Definitionen ein. Gruppenkardinalitätsintervalle in der Sprache Clafer entsprechen Gruppeninstanz-Kardinalitätsintervallen aus kardinalitätsbasierten Feature-Modellen.

Beispiel 3.1 (Schachtelungsrelationen in Clafer) In der in Listing 3.1 gezeigten Spezifikation des in dieser Arbeit betrachteten selbst-adaptiven Kommunikationssystems werden Kontext-Features des Systems repräsentiert, indem sie als Clafer-Definitionen in der Clafer-Definition `Context` geschachtelt werden. Sonstige Clafer-Definitionen der Clafer-Spezifikation repräsentieren rekonfigurierbare System-Features oder Lösungsraumkomponenten.

In Listing 3.1 ist die Clafer-Definition `connections` in der Clafer-Definition `Node` geschachtelt (siehe ❸). Aufgrund des Multiplizitätsintervalls $0..*$ darf eine Instanz der Clafer-Definition `Node` über beliebig viele Instanzen der Clafer-Definition `connection` verfügen. Das Gruppenkardinalitätsintervall $1..*$ der Clafer-Definition `Interface` (siehe ❻) legt fest, dass die Anzahl der Instanzen geschachtelter Kind-Clafer-Definition `LTE` und `Wifi` in Summe mindestens einmal pro Instanz einer Clafer-Definition `Interface` instanziiert werden muss.

Referenzen

Referenzen zu Instanzen anderer Clafer-Definition werden syntaktisch mit dem Symbol `->` zwischen Referenz-Clafer-Definition und einer Menge von Instanzen der Ziel-Clafer-Definition spezifiziert. Die Menge der Instanzen der Ziel-Clafer-Definition ist spezifiziert durch einen Mengenausdruck, der die referenzierte Instanzmenge berechnet. Jede Instanz der Referenz-Clafer-Definition muss genau eine Instanz der durch den Mengenausdruck spezifizierten Menge referenzieren. Instanzen dürfen hierbei mehrfach von verschiedenen Instanzen der Referenz-Clafer-Definition referenziert werden, falls die Instanzen der Referenz-Clafer-Definition unterschiedlichen Instanzen der Eltern-Clafer-Definition zugeordnet sind.

Beispiel 3.2 (Referenzen in Clafer) In Listing 3.1 referenziert die Referenz-Clafer-Definition `connections` (siehe ❸) eine Menge von Instanzen der Clafer-Definition `Connection`. Jede Instanz der Referenz-Clafer-Definition `connections` muss genau eine Instanz der Ziele-Clafer-Definition `Connection` referenzieren.

Jeder Instanz der Clafer-Definition `Node` sind (wegen dem Multiplizitätsintervall $0..*$) potenziell beliebig viele Instanzen der Clafer-Definition `connections` zugeordnet. Instanzen der Clafer-Definition `connections`, die der gleichen Instanz der Clafer-

Definition `Node` zugeordnet sind, dürfen nicht mehrfach die gleiche Instanz der Ziel-Clafer-Definition `Connection` referenzieren. Instanzen der Referenz-Clafer-Definition `connections`, die hingegen verschiedenen Instanzen der Eltern-Clafer-Definition `Node` zugeordnet sind, dürfen Instanzen der Ziel-Clafer-Definition `Connection` mehrfach referenzieren.

Darüber hinaus können Instanzen einer Referenz-Clafer-Definition die *primitiven Mengen* **`integer`** oder **`real`** referenzieren. Auf diese Weise können ganzzahlige oder reellwertige Attribute aus Klassendiagrammen und numerische Features repräsentiert werden. Die primitiven Mengen haben weder eine Unter- noch Obergrenze. Eine Instanz einer Referenz-Clafer-Definition, die eine primitive Menge referenziert, referenziert genau einen Wert dieser Menge. Werte der primitiven Mengen können mehrfach von Instanzen der Referenz-Clafer-Definition referenziert werden, die potentiell der gleichen Instanz ihrer Eltern-Clafer-Definition zugeordnet sind.

Beispiel 3.3 (Attribute in Clafer) In Listing 3.1 referenziert Clafer-Definition `p` die primitive Menge **`real`** (siehe 22) und repräsentiert somit ein reellwertiges Attribut. Instanzen der Clafer-Definition `p` können beliebige reelle Werte referenzieren.

Vererbungsbeziehungen

Die Sprache Clafer unterstützt nicht-überlappende und disjunkte Vererbungsbeziehungen zwischen Clafer-Definitionen. Eine Clafer-Definition kann nur von einer Clafer-Definition erben, die als *abstrakt* durch das Schlüsselwort **`abstract`** deklariert ist. Eine Instanz einer Sub-Clafer-Definition erbt Gruppenkardinalitätsintervalle, Instanzen geschachtelter Kind-Clafer-Definitionen und Constraints von ihrer in der Vererbungshierarchie übergeordneten (abstrakten) Clafer-Definition. Abstrakte Clafer-Definitionen dürfen nicht instanziiert werden und kommen somit nicht in Clafer-Spezifikationsinstanzen vor.

Beispiel 3.4 (Vererbungsbeziehungen in Clafer) In Listing 3.1 erben die Sub-Clafer-Definitionen `MobileNode` (siehe 7) und `InfrastructureNode` (siehe 14) von der abstrakten Clafer-Definition `Node` (siehe 2). Die Sub-Clafer-Definitionen `SlaveNode` (siehe 9), `MasterNode` (siehe 13) und `AutonomousNode` (siehe 10) sind wiederum von der abstrakten Clafer-Definition `MobileNode` abgeleitet. Instanzen der Clafer-Definition `MobileNode` können somit Instanzen der in der Clafer-Definition `Node` geschachtelten Kind-Clafer-Definitionen `connections` und `Interface` enthalten. Zusätzlich gelten die in der Clafer-Definition `Node` geschachtelten Constraints für Instanzen der Sub-Clafer-Definitionen.

Constraints

In Clafer charakterisieren Constraints strukturelle Einschränkungen zwischen Instanzen von Clafer-Definitionen, die nicht durch die bisher vorgestellten Sprachmittel charakterisiert werden können. Constraints haben die Ausdrucksstärke von Prädikatenlogik erster Ordnung [9]. In der Sprache Clafer kann ein Constraint aus (i) Pfadausdrücken, (ii) Mengenoperationen, (iii) Quantifizierungsausdrücken, (iv) numerischen Operationen sowie (v) aussagenlogischen Operationen bestehen.

Eine besondere Bedeutung bei der Spezifikation von Abhängigkeiten zwischen Instanzen von Clafer-Definitionen kommt Pfadausdrücken zu. Pfadausdrücke bestehen aus Bezeichnern von Clafer-Definitionen sowie den speziellen Schlüsselwörtern **`this`**, **`parent`** und **`dref`**, die syntaktisch durch Punkte voneinander getrennt sind. Durch einen Pfadausdruck wird eine Instanzmenge bzw. ein ganzzahliger oder ein reeller Wert referenziert (für Referenzen in die primitiven Mengen). Ein Pfadausdruck kann als Traversieren der Schachtelungs- bzw. Vererbungshierarchie sowie von Referenzen interpretiert werden, wobei durch jeden Traversierungsschritt Instanzen von Clafer-Definitionen zueinander in Beziehung gesetzt werden:

- Durch Bezeichner von Clafer-Definitionen wird in der Schachtelungs- bzw. Vererbungshierarchie abwärts traversiert. Instanzen einer Eltern-Clafer-Definition werden zu Instanzen der Kind-Clafer-Definition in Beziehung gesetzt. Die Kind-Clafer-Definition kann hierbei auch von einer in der Vererbungshierarchie übergeordneten Clafer-Definition geerbt sein.

- Durch das Schlüsselwort **`parent`** wird in der Schachtelungshierarchie aufwärts traversiert. Instanzen einer Kind-Clafer-Definition werden zu Instanzen der Eltern-Clafer-Definition in Beziehung gesetzt.

- Durch das Schlüsselwort **`dref`** wird von Instanzen einer Referenz-Clafer-Definition zu Instanzen der referenzierten Instanzmenge traversiert. Falls die Referenz-Clafer-Definition die primitiven Mengen **`integer`** bzw. **`real`** referenziert, bezieht sich der Pfadausdruck auf einen ganzzahligen bzw. reellen Wert.

Eine Clafer-Spezifikationsinstanz muss alle Constraints einer Spezifikation erfüllen. Ein Constraint kann entweder auf oberster Ebene außerhalb der Schachtelungshierarchie oder geschachtelt innerhalb einer Clafer-Definition spezifiziert werden. Ein Constraint, der außerhalb der Schachtelungshierarchie spezifiziert wird, bezieht sich auf alle Instanzen der Clafer-Definitionen einer Clafer-Spezifikationsinstanz. Dies entspricht einer globalen Interpretation, wie sie bei kardinalitätsbasierten Feature-Modellen vorgestellt wurde (siehe Abschnitt 2.2.4 und [105]). Derartige Constraints eignen sich somit, um globale Einschränkungen zu spezifizieren, die immer für alle Instanzen einer Clafer-Definition gelten sollen. Falls ein Constraint in einer Clafer-Definition geschachtelt ist, können außerdem lokale Einschränkungen spezifiziert werden. Die Clafer-Definition, in der

ein Constraint geschachtelt ist, wird hierbei als Kontext-Clafer-Definition bezeichnet[1]. Durch das Schlüsselwort `this` kann innerhalb von Pfadausdrücken auf die jeweilige Instanz einer Kontext-Clafer-Definition referenziert werden. Hierdurch können komplexe Abhängigkeiten zwischen Instanzen von Clafer-Definitionen charakterisiert werden.

Darüber hinaus ermöglicht Clafer aussagenlogische Ausdrücke, numerische Vergleiche, Mengenvergleiche, einfach-quantifizierende Ausdrücke über Mengen, Mengenoperationen und komplexe quantifizierte Ausdrücke mit lokalen Variablen zu spezifizieren. In nachfolgendem Beispiel demonstrieren wir exemplarisch die Semantik der genannten Constraint-Sprachausdrücke.

Beispiel 3.5 (Constraints in Clafer) In der in Listing 3.1 gezeigten Clafer-Spezifikation existieren zahlreiche Constraints, die den Raum zulässiger Clafer-Spezifikationsinstanzen zusätzlich zu Schachtelungs- und Vererbungsbeziehungen sowie Referenzen einschränken:

- Eine Kommunikationsverbindung (repräsentiert durch die Clafer-Definition `Connection`) ist charakterisiert durch einen Empfangs- und Sendeknoten (repräsentiert durch die geschachtelten Clafer-Definitionen `receiver` und `sender`) sowie dem je Verbindung konfigurierbaren Transportprotokoll (repräsentiert durch die Clafer-Definition `Protocol`). Jeder Knoten verwaltet seine aus- und eingehenden Kommunikationsverbindungen (repräsentiert durch die Clafer-Definition `connections`, die in der Clafer-Definition `Node` geschachtelt ist). Jede Kommunikationsverbindung, in der ein Knoten als Sender bzw. Empfänger fungiert, muss in der Menge aus- bzw. eingehender Kommunikationsverbindungen enthalten sein.

  Die beschriebene Konsistenzeigenschaft wird durch den Constraint 16 spezifiziert, der in der Kontext-Clafer-Definition `Connection` geschachtelt ist. Eine Instanz der Kontext-Clafer-Definition `Connection` (repräsentiert durch das Schlüsselwort `this`) muss Element der Instanzmenge sein, die durch den Pfadausdruck `this.sender.dref.connections.dref` referenziert wird. Der letztgenannte Pfadausdruck referenziert alle Verbindungen (`Connection`), die vom Sendeknoten (`this.sender`) durch die Clafer-Definition `connections` referenziert werden. Analog wird die Menge referenzierter Verbindungen für Empfangsknoten eingeschränkt.

- Der Constraint 21 limitiert die minimale und die maximale Anzahl von Knoten einer Disseminationsgruppe, die aufgrund von Ressourcenbeschränkungen (z. B. wegen Interferenz) zulässig sind. Der Kardinalitätsoperator # wertet die Anzahl der Instanzen eines Mengenausdrucks aus. Im vorliegenden Con-

1 Mit Kontext-Clafer-Definition ist *keine* Clafer-Definition gemeint, die Kontext-Features repräsentiert.

straint wird die Kardinalität der Instanzmenge, die durch den Pfadausdruck `this.members.dref.connections` referenziert wird, durch zwei konjunktiv verknüpfte Ungleichungen auf den Wertebereich $[2, 30]$ beschränkt.

- Darüber hinaus können in Clafer Operationen zwischen Mengenausdrücken spezifiziert werden. Der Constraint (12) stellt sicher, dass eine Instanz der Clafer-Definition `SlaveNode` mit genau einer Instanz der Clafer-Definition `MasterNode` über eine Instanz der Clafer-Definition `Connection` verknüpft ist. Dazu werden die Instanzmengen, die über die Pfadausdrücke `this.connections.dref.receiver.dref` und `this.connections.dref.sender.dref` referenziert werden, durch den Vereinigungsoperator `++` verknüpft. Der Schnittmengenoperator `**` reduziert die zuvor berechnete Vereinigungsmenge auf eine Instanzmenge, die ausschließlich Instanzen der Clafer-Definition `InfrastructureNode` enthält. Durch die Quantifizierungsoperation `one` wird geprüft, ob die resultierende Instanzmenge die Mächtigkeit eins hat.

  Ein weiteres Beispiel für Mengenoperationen demonstriert der Mengenausdruck (19), der von der Referenz-Clafer-Definition `members` referenziert wird: Der Mengenausdruck (mit dem Differenzmengenoperator `--`) berechnet die Differenzmenge aller Instanzen der abstrakten Clafer-Definition `MobileNode` und der konkreten Clafer-Definition `AutonomousNode`. Eine Instanz der Referenz-Clafer-Definition `members` darf somit nur Instanzen der konkreten Clafer-Definition `InfrastructureNode` und `SlaveNode` referenzieren.

- Neben einfachen Quantifizierungsausdrücken (z. B. (12) und (15)) erlaubt Clafer die Spezifikation von Constraints mit komplexen Quantifizierungsausdrücken. Der Constraint (18) stellt sicher, dass keine Instanzen der Clafer-Definition `Connection` in einer Clafer-Spezifikationsinstanz auftreten, die sowohl identische Instanzen der Clafer-Definition `Node` als Sender (`sender`) als auch identische Instanzen der Clafer-Definition `Node` als Empfänger (`receiver`) referenzieren. Dazu wird in Constraint (18) über alle disjunkten Kombinationen von Instanzen der Clafer-Definition `Connection` `c1` und `c2` quantifiziert. Für jede Kombination von `a` und `b` muss gelten, dass die jeweils durch Instanzen von Clafer-Definition `sender` und `receiver` referenzierten Instanzen der Clafer-Definition `Node` verschieden sind.

3.2 ANALYSE VON CLAFER-SPEZIFIKATIONEN

Die Sprache Clafer erlaubt die Spezifikation (globaler) Cross-Tree-Constraints aus kardinalitätsbasierten Feature-Modellen (siehe Abschnitt 2.2.3), komplexer Attribut-Constraints aus erweiterten Feature-Modellen (siehe Abschnitt 2.2.4) sowie (lokaler) Abhängigkeiten zwischen Klasseninstanzen aus OCL-Constraints in UML-Klassendiagrammen (siehe

Abschnitt 2.4). Durch die beträchtliche Ausdrucksstärke und komplexe Semantik wird jedoch die Wahrscheinlichkeit von Spezifikationsfehlern zur Entwurfszeit erhöht, die zu Fehlerzuständen des Systems zur Laufzeit führen können. Techniken zur Analyse und Validierung von Clafer-Spezifikationen sind daher erforderlich, jedoch herausfordernd (siehe Herausforderung 2). Im Folgenden beschreiben wir zunächst existierende Verfahren zur Konsistenzprüfung von Clafer-Spezifikation. Anschließend führen wir neuartige Anomalietypen ein, die in Clafer-Spezifikationen auftreten können und zur Entwurfszeit erkannt werden sollen.

3.2.1 *Konsistenzprüfung*

Die Konsistenzprüfung einer Clafer-Spezifikation kann auf folgendes Suchproblem zurückgeführt werden [6]: Falls mindestens eine Spezifikationsinstanz gefunden werden kann, die alle strukturellen Bedingungen und Constraints einer Clafer-Spezifikation erfüllt, ist die Clafer-Spezifikation konsistent.

Ähnlich wie bei UML-Objektdiagrammen (siehe Abschnitt 2.4) wird eine Spezifikationsinstanz durch eine Menge von Clafer-Definitions-Instanzen repräsentiert, die über (instanziierte) Schachtelungs- und Referenzbeziehungen verknüpft ist. Im Allgemeinen können Clafer-Spezifikationen aufgrund unbeschränkter Multiplizitätsintervalle potentiell beliebig viele Clafer-Spezifikationsinstanzen repräsentieren. Folgendes Beispiel illustriert eine mögliche Instanz der Clafer-Spezifikation, die in Listing 3.1 gezeigt ist.

```
1  InfrastructureNode
2    Interface$4
3      LTE$2
4    connections$6 -> LowLatency$2
5  SlaveNode$1
6    Interface$1
7      Wifi$1
8    connections$1 -> LowLatency$1
9    memberOf$1 -> DisseminationGroup
10 SlaveNode$2
11   Interface$2
12     Wifi$2
13   connections$2 -> Reliable
14   memberOf$2 -> DisseminationGroup
15 MasterNode
16   Interface$3
17     Wifi$3
18     LTE$1
19   connections$3 -> Reliable
20   connections$4 -> LowLatency$2
21   connections$5 -> LowLatency$1 33
22   memberOf$3 -> DisseminationGroup
23 Reliable
24   Protocol$1
25     TCP$1
26   receiver$1 -> MasterNode
27   sender$1 -> SlaveNode$2
28 LowLatency$1
29   Protocol$2
30     UDP
31   receiver$2 -> MasterNode
32   sender$2 -> SlaveNode$1
33 LowLatency$2
34   Protocol$3
35     TCP$2
36   receiver$3 -> InfrastructureNode
37   sender$3 -> MasterNode
38 DisseminationGroup 34
39   Strategy
40     Central
41   members$1 -> SlaveNode$2 35
42   members$2 -> SlaveNode$1 36
43   members$3 -> MasterNode 37
```

Listing 3.2: Instanz der Clafer-Spezifikation aus Listing 3.1 zu Systemkonfiugration C

Beispiel 3.6 (Instanz einer Clafer-Spezifikation) Abschnitt 3.2.1 zeigt eine beispielhafte Instanz der Clafer-Spezifikation aus Listing 3.1. Clafer-Definitionen, die in der Clafer-Definition `Context` geschachtelt sind, wurden hierbei aus Platzgründen nicht dargestellt. Die abgebildete Clafer-Spezifikationsinstanz repräsentiert das betrachtete selbst-adaptive Kommunikationssystem mit vier Kommunikationsknoten. Zwei Knoten fungieren als Slave-Knoten mit ausschließlich aktivierter Wi-Fi-Schnittstelle. Jeweils ein Knoten fungiert als Infrastrukturknoten (mit aktiver LTE-Schnittstelle) bzw. Relay-Knoten (repräsenteriert durch Clafer-Definition `MasterNode` und aktivierter Wi-Fi- und LTE-Schnittstelle). Innerhalb der Disseminationsgruppe wird über das Transportprotokoll UDP kommuniziert (repräsentiert durch Instanzen der Clafer-Definition `LowLatency`). Zwischen Relay-Knoten und Infrastrukturknoten wird via TCP kommuniziert (repräsentiert durch Instanzen der Clafer-Definition `Reliable`). Die Clafer-Spezifikationsinstanz entspricht somit der in Abbildung 2.1c dargestellten Systemkonfiguration C.

Konkrete Instanzen einer Clafer-Definition werden in der abgebildeten Repräsentation durch den Namen der Clafer-Definition identifiziert. Falls mehr als eine Instanz einer Clafer-Definition existiert, sind Instanzen zusätzlich nummeriert. Beispielsweise ist Clafer-Definition `connections` insgesamt fünfmal instanziiert, die fünfte Instanz der Clafer-Definition ist entsprechend `connections$5` benannt (siehe 33). Neben den Schachtelungsbeziehungen sind in der Clafer-Spezifikationsinstanz auch Referenzen zwischen Instanzen von Clafer-Definition repräsentiert. Die Instanz der Clafer-Definition `DisseminationGroup` (siehe 34) enthält beispielsweise drei Instanzen der Clafer-Definition `members` (`members$1` 35 , `members$2` 36 sowie `members$3` 37), die jeweils Instanzen der Clafer-Definition `SlaveNode` (`SlaveNode$2` und `SlaveNode$1`) und `MasterNode` referenzieren. Es können potentiell beliebig viele weitere gültige Instanzen der Spezifikation gefunden werden, da die Spezifikation Clafer-Definitionen mit unbeschränkten Multiplizitätsintervallen enthält (z. B. `Node` und `Connection`).

Existierende Verfahren zur Konsistenzprüfung von Clafer-Spezifikationen übersetzen diese entweder in die strukturelle Spezifikationssprache Alloy [76] oder verwenden Constraint-Programming-Techniken [6] zur Codierung des Suchproblems. Für die genannten Codierungen ist jedoch erforderlich, dass für Clafer-Definitionen obere Schranken der Anzahl von Instanzen manuell oder heuristisch festgelegt werden, um den Suchraum zu beschränken. Die Beschränkung des Suchraums führt allerdings zu folgender Problematik: Falls eine Clafer-Spezifikationsinstanz innerhalb des Suchraums gefunden wird, ist die Spezifikation konsistent. Falls hingegen keine gültige Clafer-Spezifikationsinstanz innerhalb des Suchraums gefunden wird, kann dies eine der folgenden Ursachen haben:

Fall 1: Die Clafer-Spezifikation ist tatsächlich inkonsistent. Somit kann, selbst wenn der Suchraum beliebig ausgedehnt wird, keine gültige Clafer-Spezifikationsinstanz gefunden werden (korrektes Ergebnis).

Fall 2: Der Suchraum ist nicht ausreichend groß gewählt. Eine passende Vergrößerung der Schranken führt somit dazu, dass eine gültige Clafer-Spezifikationsinstanz gefunden wird (falsch-negatives Ergebnis).

Bei existierenden Verfahren zur Konsistenzprüfung kann sich ein Entwickler demnach niemals sicher sein, ob der erste oder zweite Fall vorliegt. Aus diesem Grund sind bei existierenden Ansätzen langwierige, aufwändige und fehleranfällige Iterationsschleifen notwendig, in denen sukzessive der Suchraum ausgedehnt und die Instanzsuche wiederholt wird. Falls die Schranken des Suchraums viel zu groß gewählt werden, reduziert sich zudem die Effizienz der Instanzsuche, da ein entsprechend großer Suchraum inspiziert werden muss. In diesem Sinne sind existierende Analysetechniken nicht vollständig, da, falls keine gültige Instanz gefunden werden kann, die Fälle 1 und 2 nicht unterscheidbar sind (siehe Herausforderung 4).

3.2.2 *Anomaliedetektion*

Anomalien können auf Inkonsistenzen, Irregularitäten oder sonstige Abweichungen von Erwartungen hinsichtlich Form und Funktion hindeuten (siehe Abschnitt 2.2.5 und [74]). Für Clafer-Spezifikationen wurden in der Literatur über Konsistenzprüfungen hinausgehende Analyseverfahren zur Identifikation von Anomalien nicht untersucht. Wir charakterisieren nun folgende neuartigen Anomalietypen, die ursprünglich von Weckesser et al. in [149, 150] für kardinalitätsbasierte Feature-Modelle vorgeschlagen und in [148] auf Clafer-Spezifikationen übertragen wurden:

- Eine Anomalie vom Typ *tote Clafer-Definition* (engl. dead clafer) und *Kern-Clafer-Definition* (engl. core clafer) kann bei Clafer-Definitionen auftreten.
- Anomalien vom Typ *falsche Intervalluntergrenze* (engl. false lower bound) und *falsche Intervallobergrenze* (engl. false upper bound) können bei Multiplizitäts- und Gruppenkardinalitätsintervallen von Clafer-Definitionen auftreten.
- Anomalien vom Typ *fälschlicherweise unbeschränkt* (engl. falsely unbounded) und *lückenhafter Wertebereich* (engl. multiplicity gap) können bei Multiplizitätsintervallen, Gruppenkardinalitätsintervallen sowie bei Clafer-Definitionen auftreten, die ganzzahlige oder reellwertige Attribute repräsentieren.

Die in Abschnitt 3.1 präsentierte Clafer-Spezifikation des in dieser Arbeit betrachteten selbst-adaptiven Kommunikationssystems enthält eine Reihe von Anomalien, die auf Spezifikationsfehler hindeuten und zur Laufzeit des Systems potentiell zu Ausfällen oder undefinierten Zuständen führen. Im Folgenden charakterisieren wir die genannten Anomalietypen anhand des Beispiels.

Analyse von Multiplizitätsintervallen

Anomalien können bei Multiplizitätsintervallen von Clafer-Definitionen auftreten. In Abschnitt 3.1 wurde beschrieben, dass Multiplizitätsintervalle angeben, wie viele Instanzen einer Clafer-Definition mindestens oder höchstens je Instanz einer übergeordneten Clafer-Definition vorkommen dürfen. Intervallgrenzen von Multiplizitätsintervallen können durch komplexe Abhängigkeiten von anderen Clafer-Definitionen oder Constraints eingeschränkt werden. Die syntaktisch spezifizierten und die tatsächlich realisierbaren Belegungen eines Multiplizitätsintervalls weichen daher häufig voneinander ab. Falls das Multiplizitätsintervall zu breit spezifiziert wird, existiert für ein unteres oder oberes Teilintervall keine Clafer-Spezifikationsinstanz mit der geforderten Anzahl von Instanzen der Clafer-Definition. Zusätzlich vergrößert sich der Suchraum, um gültige Spezifikationsinstanzen zu identifizieren. Dies kann die Effizienz existierender Verfahren zur Konsistenzprüfung negativ beeinflussen. Umgekehrt, falls das Multiplizitätsintervall einer Clafer-Definition zu eng spezifiziert wird, kann keine Instanz der Clafer-Spezifikation gefunden werden, in der Instanzen der Clafer-Definition vorkommen.

Eine Clafer-Definition mit einem Multiplizitätsintervall, für das keine gültige Spezifikationsinstanz gefunden werden kann, wird in Anlehnung an die in Abschnitt 2.2.5 genannten Anomalietypen als *tote Clafer-Definition* bezeichnet.

Beispiel 3.7 (Anomalie vom Typ tote Clafer-Definition) In der in Listing 3.1 gezeigten Spezifikation steht die Clafer-Definition `AutonomousNode` in einer Vererbungsbeziehung zu der Clafer-Definition `MobileNode`. Die Clafer-Definition `MobileNode` enthält die geschachtelte Clafer-Definition `memberOf` mit dem Multiplizitätsintervall 0..1. Instanzen der Clafer-Definition `AutonomousNode` enthalten somit ebenfalls potentiell geschachtelte Instanzen der Clafer-Definition `memberOf`. Im vorliegenden Fall ist aufgrund des Mengenausdrucks ⓳ sichergestellt, dass eine Instanz der Clafer-Definition `memberOf` keine Instanz der Clafer-Definition `AutonomousNode` referenziert. Jedoch wird durch den Constraint ❽ gefordert, dass eine Instanz der Clafer-Definition `AutonomousNode`, die eine Instanz der Clafer-Definition `DisseminationGroup` referenziert, ebenfalls von dieser Instanz referenziert werden muss. Es existiert somit keine Clafer-Spezifikationsinstanz, in der eine Instanz der Clafer-Definition `AutonomousNode` eine geschachtelte Instanz der Clafer-Definition `memberOf` enthält. Die Clafer-Definition `memberOf` (die durch Vererbung in Clafer-Definition `AutonomousNode` geschachtelt ist) weist daher eine Anomalie vom Typ *tote Clafer-Definition* auf.

In Abschnitt 2.2.5 wurde für Feature-Modelle die Identifikation von Kern-Features als wichtiges Analyseziel vorgestellt. Analog wird eine Clafer-Definition als *Kern-Clafer-Definition* charakterisiert, falls in jeder Clafer-Spezifikationsinstanz mindestens eine Instanz der Clafer-Definition auftritt.

Beispiel 3.8 (Anomalie vom Typ Kern-Clafer-Definition) In der in Listing 3.1 gezeigten Clafer-Spezifikation haben die Clafer-Definitionen `numberOfNodes` und `Context` jeweils das Multiplizitätsintervall 1..1. Die Clafer-Definition `numberOfNodes` ist zudem in der Clafer-Definition `Context` geschachtelt. In jeder Clafer-Spezifikationsinstanz muss somit jeweils genau eine Instanz der Clafer-Definitionen `Context` und `numberOfNodes` auftreten. Bei beiden genannten Clafer-Definitionen handelt es sich daher um Kern-Clafer-Definitionen.

Anomalien können bei Multiplizitätsintervallen von Clafer-Definitionen zusätzlich an der unteren und an der oberen Grenze sowie innerhalb eines Teilintervalls auftreten. Ein Multiplizitätsintervall einer Clafer-Definition weist eine Anomalie vom Typ *falsche Intervalluntergrenze* (engl. false lower bound) auf, falls die untere Grenze des Multiplizitätsintervalls zu klein gewählt ist. In diesem Fall kann für ein unteres Teilintervall keine Spezifikationsinstanz mit der geforderten Anzahl von Instanzen der Clafer-Definition gefunden werden.

Beispiel 3.9 (Anomalie vom Typ falsche Intervalluntergrenze) In der in Listing 3.1 gezeigten Clafer-Spezifikation ist die Clafer-Definition `members` (siehe 19) mit dem Multiplizitätsintervall 0..∗ annotiert. Aufgrund des Constraints 20 wird jedoch gefordert, dass eine Instanz der Clafer-Definition `DisseminationGroup` mindestens eine Instanz der Clafer-Definition `MasterNode` referenziert. Die tatsächliche untere Grenze des Multiplizitätsintervalls ist daher 1. Das Multiplizitätsintervall der Clafer-Definition `members` weist somit eine Anomalie vom Typ *falsche Intervalluntergrenze* auf.

Ein Multiplizitätsintervall einer Clafer-Definition weist eine Anomalie vom Typ *falsche Intervallobergrenze* (engl. false upper bound) auf, falls die obere Grenze des Multiplizitätsintervalls so gewählt ist, dass für ein Teilintervall an der oberen Grenze des Multiplizitätsintervalls keine Spezifikationsinstanz mit der geforderten Anzahl von Instanzen der Clafer-Definition gefunden werden kann.

Beispiel 3.10 (Anomalie vom Typ falsche Intervallobergrenze) In der in Listing 3.1 gezeigten Clafer-Spezifikation ist das Multiplizitätsintervall 0...∗ der Clafer-Definition `members` (siehe 19) zusätzlich nach oben hin eingeschränkt durch den Constraint 21. Die Anzahl der referenzierten Instanzen der Clafer-Definition `Connection` ist somit auf 30 limitiert. Instanzen der Clafer-Definition `members` referenzieren Instanzen der Clafer-Definition `MobileNode`. Diese enthalten wiederum mindestens eine geschachtelte Instanz der Clafer-Definition `connections`. Die tatsächliche obere Grenze des Multiplizitätsintervalls von `members` ist beschränkt auf 30 und weist dementsprechend eine Anomalie vom Typ *falsche Intervallobergrenze* auf.

Eine wichtiges Analyseziel für (syntaktisch) unbeschränkte Multiplizitätsintervalle $lb..*$ ist es, die tatsächliche (semantische) Unbeschränktheit festzustellen. Das Multiplizitätsintervall einer Clafer-Definition ist *fälschlicherweise unbeschränkt* (engl. falsely unbounded), falls als obere Grenze das Symbol $*$ angegeben ist und das Multiplizitätsintervall zudem eine Anomalie vom Typ *falsche Intervallobergrenze* aufweist. Entsprechend ist das Multiplizitätsintervall einer Clafer-Definition *tatsächlich unbeschränkt*, falls als obere Grenze des Multiplizitätsintervalls das Symbol $*$ angegeben ist und Clafer-Spezifikationsinstanzen mit potentiell beliebig vielen Instanzen der Clafer-Definition existieren.

Beispiel 3.11 (Tatsächliche Unbeschränktheit eines Multiplizitätsintervalls) In der in Listing 3.1 gezeigten Spezifikation ist die Clafer-Definition `AutonomousNode` mit dem Multiplizitätsintervall $0..*$ spezifiziert (siehe ❿). Die Anzahl der Instanzen der Clafer-Definition `AutonomousNode` ist weder durch Constraints noch durch Abhängigkeiten zu anderen Clafer-Definitionen eingeschränkt. Es können somit Clafer-Spezifikationsinstanzen mit potentiell beliebig vielen Instanzen der Clafer-Definition `AutonomousNode` gefunden werden. Das Multiplizitätsintervall ist demnach tatsächlich unbeschränkt. Entsprechend existieren somit potentiell beliebig viele Clafer-Spezifikationsinstanzen.

Darüber hinaus können Anomalien auch innerhalb eines gültigen Teilbereichs des Multiplizitätsintervalls einer Clafer-Definition auftreten. Ein Multiplizitätsintervall einer Clafer-Definition weist eine Anomalie vom Typ *lückenhafter Wertebereich* (engl. multiplicity gap) auf, falls für ein Teilintervall, das nicht an den Intervallgrenzen liegt, keine Spezifikationsinstanz mit der geforderten Anzahl von Instanzen der Clafer-Definition gefunden werden kann.

Beispiel 3.12 (Anomalie vom Typ lückenhafter Wertebereich) In der in Listing 3.1 gezeigten Spezifikation ist die Clafer-Definition `connections` mit dem Multiplizitätsintervall $0..*$ annotiert (siehe ❸) und in der Clafer-Definition `Node` geschachtelt. Aufgrund des Constraints ❹ dürfen (abgeleitete) Instanzen der abstrakten Clafer-Definition `Node` jedoch nicht 3 oder 4 geschachtelte Instanzen der Clafer-Definition `connections` enthalten. Für das Teilintervall $3..4$ des Multiplizitätsintervalls können somit keine gültigen Spezifikationsinstanzen gefunden werden. Es handelt sich daher um eine Anomalie vom Typ *lückenhafter Wertebereich*. Da in der Sprache Clafer keine zusammengesetzten Intervalle in der Form $lb..ub, lb..ub$ spezifiziert werden können, müssen identifizierte Anomalien vom Typ *lückenhafter Wertebereich* beispielsweise durch zusätzliche Constraints explizit sichtbar gemacht werden.

Die Analyse von Multiplizitätsintervallen von Clafer-Definitionen wird dazu eingesetzt, um zur Entwurfszeit Anomalien zu identifizieren. Darüber hinaus liefert die Analyse Informationen über die Häufigkeit des maximalen Vorkommens von Instanzen einer Clafer-Definition.

Analyse von Gruppenkardinalitätsintervallen

In Gruppenkardinalitätsintervallen von Clafer-Definitionen können Anomalien auftreten, die auf Spezifikationsfehler hindeuten oder zu inkonsistenten Clafer-Spezifikationen führen. Wie bei der Analyse von Multiplizitätsintervallen von Clafer-Definitionen können Anomalien an der Unter- und Obergrenze sowie innerhalb von Teilintervallen auftreten. Für Gruppenkardinalitätsintervalle mit Anomalien vom Typ *falsche Intervalluntergrenze* und *falsche Intervallobergrenze* können semantisch äquivalente und anomaliefreie Ersetzungen angegeben werden.

Beispiel 3.13 (Anomalie an den Grenzen eines Gruppenkardinalitätsintervalls) In Listing 3.1 ist die Clafer-Definition `Context` mit dem Gruppenkardinalitätsintervall $0..*$ annotiert (siehe 24). Da immer genau eine Instanz der Clafer-Definition `numberOfNodes` (siehe 25) pro Instanz der Clafer-Definition `Context` existieren muss, ist die tatsächliche untere Grenze des Gruppenkardinalitätsintervalls gleich eins. Pro Instanz der Clafer-Definition `Context` können höchstens drei geschachtelte Clafer-Definitionen instanziiert sein. Das Gruppenkardinalitätsintervall weist somit sowohl eine Anomalie vom Typ *falsche Intervalluntergrenze* als auch eine Anomalie vom Typ *falsche Intervallobergrenze* auf. Aufgrund des Constraints 32 muss, falls die Clafer-Definition `Mode` mindestens einmal instanziiert ist, ebenfalls die Clafer-Definition `mobility` mindestens einmal instanziiert sein. In Summe sind in einer Instanz der Clafer-Definition `Context` entweder eine oder drei Instanzen anderer Clafer-Definitionen geschachtelt. Das Gruppenkardinalitätsintervall weist daher für das Teilintervall 2..2 eine Anomalie vom Typ *lückenhafter Wertebereich* auf.

Analyse ganzzahliger und reellwertiger Attribute

In Clafer-Spezifikationen werden ganzzahlige und reellwertige Attribute durch Clafer-Definitionen repräsentiert, welche die Mengen `integer` oder `real` referenzieren. Der zulässige Wertebereich der numerischen Attribute wird durch zusätzliche Constraints eingeschränkt. Derartige Einschränkungen der Wertebereiche sind meist nicht offensichtlich und erfolgen häufig unbeabsichtigt. Zudem müssen für die Konsistenzprüfung von Clafer-Spezifikationen obere Schranken für ganzzahlige und reelle Werte angegeben werden (siehe Abschnitt 3.1). Die Identifikation des tatsächlich gültigen Wertebereichs einer Clafer-Definition, die ein Attribut repräsentiert, ist daher ein wichtiges Analyseziel.

Eine Clafer-Definition, die ein Attribut repräsentiert, weist eine Anomalie vom Typ *fälschlicherweise unbeschränkt* (engl. falsely unbounded) auf, falls der Wertebereich des Attributs nach oben oder unten beschränkt ist. Demgegenüber ist ein Attribut *tatsächlich nach oben oder unten unbeschränkt*, falls dessen Wertebereich durch keinen Constraint beschränkt ist.

Beispiel 3.14 (Unbeschränktheit des Wertebereichs eines Attributs) In der in Listing 3.1 gezeigten Spezifikation referenziert der Clafer-Definition `numberOfNodes` die primitive Menge **`integer`** und repräsentiert somit ein ganzzahliges Attribut (siehe 25 in Listing 3.1). Durch den Constraint 26 wird einer Instanz der Clafer-Definition `numberOfNodes` als Wert die Anzahl der (abgeleiteten) Instanzen von Clafer-Definition `Node` zugewiesen. Da die Anzahl von Instanzen einer Clafer-Definition in einer Clafer-Spezifikationsinstanz nicht negativ sein kann, ist der Wertebereich des Attributs `numberOfNodes` nach unten hin beschränkt. Der Wertebereich des Attributs `numberOfNodes` ist hingegen nach oben hin unbeschränkt, da Clafer-Spezifikationsinstanzen mit potentiell beliebig vielen Instanzen der Clafer-Definition `Node` existieren.

Für Clafer-Definitionen, die Attribute repräsentieren, können darüber hinaus Teile des Wertebereichs existieren, für die keine gültige Spezifikationsinstanz gefunden werden können. Analog zur Analyse von Multiplizitäts- und Gruppenkardinalitätsintervallen von Clafer-Definitionen wird dies als Anomalie vom Typ *ungültiger Wertebereich* bezeichnet. Derartige Anomalien sind häufig durch explizite Wertzuweisungen in Constraints beabsichtigt, können jedoch auch auf Spezifikationsfehler hindeuten.

Beispiel 3.15 (Beschränktheit des Wertebereichs eines Attributs) In der in Listing 3.1 gezeigten Spezifikation repräsentiert die Clafer-Definition `p` ein reellwertiges Attribut (siehe 22). Durch den Constraint 23 muss das Attribut `p` größer oder gleich null und kleiner oder gleich eins sein. Zusätzlich darf `p` nicht den Wert $0,5$ annehmen. Das Attribut `p` ist somit sowohl nach oben als auch nach unten beschränkt und weist für den Wert $0,5$ eine Anomalie vom Typ *ungültiger Wertebereich* auf.

Analyse der Kontextvariabilität

Weitere Analyseziele ergeben sich bei der Analyse von Spezifikationen der Kontextvariabilität in Form von Kontext-Feature-Modellen, die in Abschnitt 2.3 vorgestellt wurden. Die Planung mittels Kontext-Feature-Modellen zur Laufzeit setzt voraus, dass zu jeder Kontext-Konfiguration mindestens eine konsistente Systemkonfiguration existiert. Die Identifikation von Kontextkonfigurationen zur Entwurfszeit, für die keine Systemkonfiguration existiert, ist somit ein wichtiges Analyseziel (siehe Herausforderungen 1 und 2). Die zuvor vorgestellten Analysen von Multiplizitätsintervallen können verwendet werden, um Inkonsistenzen in den Abhängigkeiten zwischen Kontext- und Systemelementen zu identifizieren. Für eine Clafer-Definition, die ein Kontext-Feature repräsentiert, existiert keine konsistente Systemkonfiguration, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Clafer-Definition weist eine Anomalie vom Typ *tote Clafer-Definition* auf.

- Falls mindestens eine Instanz der Clafer-Definition in einer Clafer-Spezifikationsinstanz vorkommt und dies impliziert, dass alle Clafer-Definitionen, die System-Features oder Lösungsraumkomponenten repräsentieren, eine Anomalie vom Typ *tote Clafer-Definition* aufweisen.

Folgendes Beispiel demonstriert die Analyse einer Clafer-Definition, die ein Kontext-Feature repräsentiert.

Beispiel 3.16 (Analyse von Kontext-Feature repräsentierenden Clafer-Definitionen) In der in Listing 3.1 gezeigten Clafer-Spezifikation erben System-Features und rekonfigurierbare Elemente des Lösungsraums, die sich auf höchster Ebene der Schachtelungshierarchie befinden, von der abstrakten Clafer-Definition `SystemElement`. Falls mindestens eine Instanz der Clafer-Definition `SystemElement` in einer Clafer-Spezifikationsinstanz existiert, kann geschlossen werden, dass für die rekonfigurierbaren Teile des selbst-adaptiven Kommunikationssystems ein konsistenter Systemzustand existiert.

In der in Listing 3.1 gezeigten Clafer-Spezifikation repräsentiert die Clafer-Definition `Normal` ein Kontext-Feature (siehe 30). Es existiert jedoch keine Clafer-Spezifikationsinstanz, in der Instanzen der Clafer-Definition `Normal` auftreten. Die Clafer-Definition `Normal` weist daher eine Anomalie vom Typ *tote Clafer-Definition* auf. Es kann somit ausgeschlossen werden, dass konsistente Systemzustände berechnet werden können, falls das genannte Kontext-Feature gewählt ist.

Des Weiteren kann eine Clafer-Definition, die ein Kontext-Feature repräsentiert, als redundant charakterisiert werden, falls sie keine Abhängigkeiten zu rekonfigurierbaren Teilen des Systems aufweist. Die An- oder Abwahl redundanter Kontext-Features durch die Systemumgebung hat in diesem Fall keine Auswirkung auf die Konfigurationsoptionen des Systems. Folgendes Beispiel demonstriert eine Clafer-Definition, die ein redundantes Kontext-Feature repräsentiert.

Beispiel 3.17 (Analyse von redundanten Kontext-Feature repräsentierenden Clafer-Definitionen) In der in Listing 3.1 gezeigten Clafer-Spezifikation repräsentiert die Clafer-Definition `mobility` ein Kontext-Feature (siehe 30). Falls mindestens eine Instanz der Clafer-Definition `mobility` in einer Clafer-Spezifikationsinstanz auftritt, hat dies keine Auswirkung auf Instanzen von Clafer-Definitionen, die System-Features oder Komponenten der Lösungsraumarchitektur repräsentieren, da zwischen der Clafer-Definition `mobility` und anderen Elementen der Spezifikation keine Abhängigkeiten existieren. Die Clafer-Definition `mobility` repräsentiert somit ein redundantes Kontext-Feature im Problemraum.

4

FORMALES RAHMENWERK

In diesem Kapitel führen wir das formale Rahmenwerk ein, das im Rahmen dieser Arbeit verwendet wird. Zunächst beschreiben wir in Abschnitt 4.1 die abstrakte Syntax von Clafer-Spezifikationen. Im zweiten Schritt definieren wir in Abschnitt 4.2 Wohlgeformtheitseigenschaften, die für syntaktisch korrekte Clafer-Spezifikationen erfüllt sein müssen. In Abschnitt 4.3 charakterisieren wir die Semantik von Clafer-Spezifikationen und definieren semantische Eigenschaften sowie Anforderungen an das in dieser Arbeit präsentierte Analyseverfahren.

4.1 SYNTAX VON CLAFER-SPEZIFIKATIONEN

Im Folgenden definieren wir die abstrakte Syntax von Clafer-Spezifikationen. Dies erlaubt es uns, die semantischen Eigenschaften des in dieser Arbeit präsentierten Analyseverfahrens zu charakterisieren. Zunächst führen wir die abstrakte Syntax für Multiplizitätsintervalle in der Form (l, u) als Paar von unterer und oberer Schranke ein. Sowohl die untere als auch die obere Schranke eines Multiplizitätsintervalls wird durch eine natürliche Zahl festgelegt. Für die obere Schranke ist zusätzlich das spezielle Symbol $*$ zulässig, das unbeschränkte Multiplizitätsintervalle kennzeichnet. Per Konvention gilt für beliebige $k \in \mathbb{N}_0$ die Bedingung $k < *$.

> **Beispiel 4.1** (Abstrakte Syntax der Intervallsprache) Das Multiplizitätsintervall 1..$*$ wird in der abstrakten Syntax durch das Paar $(1, *)$ repräsentiert. Das Multiplizitätsintervall ist unbeschränkt, da für die obere Schranke das spezielle Symbol $*$ verwendet wird.

Ein Multiplizitätsintervall lässt sich somit folgendermaßen definieren:

> **Definition 4.2** (Multiplizitätsintervall) Ein Multiplizitätsintervall ist ein Paar (l, u) mit der unteren Schranke $l \in \mathbb{N}_0$ und der oberen Schranke $u \in (\mathbb{N}_0 \cup \{*\})$. Ein Multiplizitätsintervall ist *syntaktisch unbeschränkt*, falls $u = *$ gilt. Die Menge $\mathcal{I} \subset \mathbb{N}_0 \times (\mathbb{N}_0 \cup \{*\})$ enthält alle Multiplizitätsintervalle $(l, u) \in \mathcal{I}$, für welche die Bedingung $l \leq u$ gilt.

Eine Clafer-Spezifikation besteht aus einer endlichen Menge C von Clafer-Definitionen. Einer Clafer-Definition $c \in C$ ist über die Bezeichnerfunktion $id : C \to \langle identifier \rangle$ ein Bezeichner zugeordnet. Die Schachtelungshierarchie zwischen Clafer-Definitionen ist durch die Relation $\blacklozenge\!\!\rightarrow$ auf der Menge C definiert, d. h. es gilt: $\blacklozenge\!\!\rightarrow \in C \times C$. Wir schreiben $c \blacklozenge\!\!\rightarrow c'$, wenn c' in c geschachtelt ist. Der Bezeichner $\mathrm{id}(c')$ einer Clafer-Definition c' muss eindeutig sein für alle Clafer-Definitionen, die in der Eltern-Clafer-Definition c geschachtelt sind. Die Menge C besteht aus den disjunkten Teilmengen abstrakter Clafer-Definitionen C_A und konkreter Clafer-Definitionen C_O, sodass die Bedingung $C = C_A \cup C_O$ gilt. Die konkrete Hilfs-Clafer-Definition $c_r \in C_O$ dient als Wurzel der Schachtelungs- und Vererbungshierarchie. Es muss somit für alle Clafer-Definitionen c, die sich auf oberster Ebene der Schachtelungshierarchie befinden, $c_r \blacklozenge\!\!\rightarrow c$ gelten.

Beispiel 4.3 (Abstrakte Syntax von Schachtelungsrelationen) Für die in Listing 3.1 gezeigte Spezifikation ist die Menge aller Clafer-Definitionen definiert als $C = \{$`Node`, `connections`, `Interface`, `LTE`, `Wifi`, ...$\}$. Die Clafer-Definitionen `connections` und `Interface` sind in der Clafer-Definition `Node` geschachtelt. In der abstrakten Syntax existieren somit die Relationen `Node` $\blacklozenge\!\!\rightarrow$ `connections` und `Node` $\blacklozenge\!\!\rightarrow$ `Interface`. Die Clafer-Definition `Node` befindet sich auf der obersten Ebene der Schachtelungshierarchie. Demnach existiert die Relation $c_r \blacklozenge\!\!\rightarrow$ `Node`. Da die Clafer-Definition `Node` abstrakt ist, gilt `Node` $\in C_A$. Entsprechend, da die Clafer-Definition `Interface` konkret ist, gilt `Interface` $\in C_O$.

Die Vererbungshierarchie von Clafer-Definitionen wird durch die Relation $\twoheadleftarrow$ definiert. Wir schreiben $c \twoheadleftarrow c'$, wenn eine Clafer-Definition $c' \in C$ geschachtelte Clafer-Definitionen, Referenzen sowie Constraints seiner in der Vererbungshierarchie übergeordneten abstrakten Clafer-Definition $c \in C_A$ erbt. Die abstrakte Hilfs-Clafer-Definition $c_a \in C_A$ dient als Wurzelknoten der Vererbungshierarchie. Alle abstrakten Clafer-Definitionen, die über keine explizit spezifizierte und in der Vererbungshierarchie übergeordnete abstrakte Clafer-Definition verfügen, erben von c_a. Die Vererbungsrelation bildet einen endlichen Wurzelbaum auf der Menge $C \cup C'_O$, wobei die Menge $C'_O = \{\, c \in C_O \mid c' \in C_A \wedge c' \twoheadleftarrow c \,\}$ alle konkreten Clafer-Definitionen enthält, die von abstrakten Clafer-Definitionen erben. Die Vererbungsrelation wird zusätzlich durch die Schachtelungshierarchie eingeschränkt. Geschachtelte Clafer-Definitionen dürfen nur von abstrakten Clafer-Definitionen erben, die über identische Eltern-Clafer-Definitionen verfügen oder die sich in der darüber liegenden Schachtelungshierarchie befinden.

Beispiel 4.4 (Abstrakte Syntax von Vererbungsrelationen) In der in Listing 3.1 gezeigten Clafer-Spezifikation sind die Clafer-Definitionen `Node` und `MobileNode` abstrakt. Die Clafer-Definition `MasterNode` ist konkret. Die Clafer-Definition `Node` ist auf der obersten Ebene der Vererbungshierarchie. Es existiert somit in der abstrakten Syntax die Relation $c_a \twoheadleftarrow$ `Node`. Die Clafer-Definition `MobileNode` erbt von der Clafer-Defini-

tion `Node`. Die Clafer-Definition `MasterNode` erbt wiederum von der Clafer-Definition `MobileNode`. Es existieren somit zusätzlich die Relationen `Node` $\leftarrow$ `MobileNode` und `MobileNode` $\leftarrow$ `MasterNode`.

Jeder Clafer-Definition $c \in C$ ist ein Multiplizitätsintervall durch die Funktion $\lambda_c(c)$ sowie ein Gruppenkardinalitätsintervall durch die Funktion $\lambda_g(c)$ zugewiesen. Die Funktion $\lambda_g(c)$ legt das Kardinalitätsintervall der Menge aller Sub-Clafer-Definitionen von c in Bezug auf die Schachtelungsrelation $\blacklozenge\!\!\rightarrow$ fest.

Beispiel 4.5 (Abstrakte Syntax von Multiplizitäts- und Gruppenkardinalitätsintervallen) Für die in Listing 3.1 gezeigte Clafer-Spezifikation ist der Clafer-Definition `Interface` das Multiplizitätsintervall 1..1 sowie das Gruppenkardinalitätsintervall 1..$*$ zugeordnet. In der abstrakten Syntax ergibt sich somit $\lambda_c(\texttt{Interface}) = (1,1)$ und $\lambda_g(\texttt{Interface}) = (1,*)$.

Eine Referenz von einer Clafer-Definition zu einer Menge von Clafer-Definitions-Instanzen wird durch die Relation $\twoheadrightarrow\!\in\ c \times \langle setExpr \rangle$ repräsentiert. Die referenzierte Menge wird durch einen Mengenausdruck `sExp` $\in \langle setExpr \rangle$ über Elemente von C spezifiziert. Für Clafer-Definitionen, welche die primitive Menge **`integer`** referenzieren, führen wir spezielle Clafer-Definitionen ein, die ganzzahlige Werte repräsentieren. Die eingeführten speziellen Clafer-Definitionen sind Elemente der Menge $C_{\mathbb{Z}} := \{c_z \mid z \in \mathbb{Z}\}$. Eine Clafer-Definition c, die einen ganzzahligen Wert z referenziert, wird somit durch die Relation $c \twoheadrightarrow c_z$ mit $c_z \in C_{\mathbb{Z}}$ repräsentiert. Referenzen zu reellen Werten der primitive Menge **`real`** werden entsprechend durch Relationen zu Clafer-Definitionen der Menge $C_{\mathbb{R}} := \{c_i \mid i \in \mathbb{R}\}$ repräsentiert.

Beispiel 4.6 (Abstrakte Syntax von Referenzen) In der in Listing 3.1 gezeigten Clafer-Spezifikation referenziert die Clafer-Definition `members` die Mengendifferenz `sExp` := `MobileNode -- AutonomousNode` (siehe 19 in Listing 3.1). In der abstrakten Syntax existiert somit die Relation `members` $\twoheadrightarrow$ `sExp`. Die Clafer-Definition `p` referenziert die primitive Menge **`real`** und repräsentiert somit ein reellwertiges Attribut (siehe 22 in Listing 3.1). In der abstrakten Syntax werden diese Referenzen repräsentiert durch eine Relation `p` $\twoheadrightarrow c_z$ mit $c_z \in C_{\mathbb{R}}$.

Ein Constraint wird in der abstrakten Syntax durch einen booleschen Ausdruck der Form `bExp` $\in \langle boolExpr \rangle$ repräsentiert. Die in Abbildung 4.1 gezeigte Grammatik beschreibt die Syntax boolescher Ausdrücke. Die Schachtelung von Constraints in ihren Kontext-Clafer-Definitionen wird durch die Relation $\Phi \in C \times \langle boolExpr \rangle$ repräsentiert. Wir schreiben $\Phi(c, \texttt{bExp})$, falls der Ausdruck `bExp` in Kontext-Clafer-Definition c geschachtelt ist. Für alle Constraints `bExp`$'$, die nicht geschachtelt sind und somit global gelten müssen, fungiert die konkrete Wurzel-Clafer-Definition c_r als Kontext-Clafer-Definition. Die Relation $\Phi(c_r, \texttt{bExp}')$ muss für globale Constraints folglich existieren.

Beispiel 4.7 (Abstrakte Syntax von Constraints) In der in Listing 3.1 gezeigten Clafer-Spezifikation wird die Constraint ❹ durch den aussagenlogischen Ausdruck `bExp` und die Constraint ⓲ durch `bExp'` repräsentiert. Der Ausdruck `bExp` ist in der Clafer-Definition `connections` geschachtelt. Die Clafer-Definition `connections` fungiert daher als Kontext-Clafer-Definition von `bExp`. In der abstrakten Syntax existiert somit die Relation $\Phi(\texttt{connections}, \texttt{bExp})$. Der Ausdruck `bExp'` befindet sich auf der obersten Ebene der Schachtelungshierarchie. Die Clafer-Definition c_r fungiert demnach als Kontext-Clafer-Definition von `bExp'`. In der abstrakten Syntax existiert folglich die Relation $\Phi(c_r, \texttt{bExp}')$.

Mit der eingeführten abstrakten Syntax lässt sich eine Clafer-Spezifikation folgendermaßen definieren:

Definition 4.8 (Clafer-Spezifikation) Eine Clafer-Spezifikation ist definiert als Tupel $cs = (C, \langle identifier \rangle, \mathrm{id}, C_A, c_r, c_a, \leftarrowtriangle, \blacklozenge\!\!\rightarrow, \lambda_c, \lambda_g, \langle setExpr \rangle, \twoheadrightarrow, \langle boolExpr \rangle, \Phi)$ über der nicht-leeren und endlichen Menge von Clafer-Definitionen C. Die Sprache Clafer ist definiert als die Menge aller syntaktisch wohlgeformten Clafer-Spezifikationen CS mit $cs \in CS$. Die Attribute des Tupels cs sind wie folgt definiert:

- $\langle identifier \rangle$ ist die Menge aller *Bezeichner* gemäß der Grammatik in Abbildung 4.1,
- $\mathrm{id} : C \to \langle identifier \rangle$ ist die *Bezeichnerfunktion* einer Clafer-Definition,
- $C_A \subseteq C$ ist die Menge aller abstrakten Clafer-Definitionen,
- $C_O \subseteq C \setminus C_A$ ist die Menge aller konkreten Clafer-Definitionen,
- $c_r \in C_O$ ist die konkrete Wurzel-Clafer-Definition,
- $c_a \in C_A$ ist die abstrakte Wurzel-Clafer-Definition,
- $\leftarrowtriangle \subseteq C_A \times C$ ist die *Vererbungsrelation*, $\leftarrowtriangle^+$ bezeichnet die transitive Hülle und $\leftarrowtriangle^*$ bezeichnet die reflexiv-transitive Hülle von $\leftarrowtriangle$,
- $\blacklozenge\!\!\rightarrow \subseteq C \times C$ ist die *Schachtelungsrelation*, $\blacklozenge\!\!\rightarrow^+$ bezeichnet die transitive Hülle und $\blacklozenge\!\!\rightarrow^*$ bezeichnet die reflexiv-transitive Hülle von $\blacklozenge\!\!\rightarrow$,
- $\lambda_c : C \to \mathcal{I}$ ist die *Multiplizitätsfunktion*,
- $\lambda_g : C \to \mathcal{I}$ ist die *Gruppenkardinalitätsfunktion*,
- $\langle setExpr \rangle$ ist die Menge aller *Mengenausdrücke*, die den Regeln der Grammatik in Abbildung 4.1 folgen.

⟨*boolExpr*⟩ ::= '(' ⟨*boolExpr*⟩ ')'
| '!' ⟨*boolExpr*⟩
| ⟨*boolExpr*⟩ ('&&' ⟨*boolExpr*⟩)+
| ⟨*boolExpr*⟩ ('||' ⟨*boolExpr*⟩)+
| ⟨*numericComp*⟩
| ⟨*setQuantSimple*⟩
| ⟨*setQuant*⟩
| ⟨*setComp*⟩

⟨*setComp*⟩ ::= ⟨*setExpr*⟩ 'in' ⟨*setExpr*⟩
| ⟨*setExpr*⟩ 'not in' ⟨*setExpr*⟩
| ⟨*setExpr*⟩ '=' ⟨*setExpr*⟩
| ⟨*setExpr*⟩ '!=' ⟨*setExpr*⟩

⟨*setQuant*⟩ ::= 'lone' ⟨*setExpr*⟩ | 'one' ⟨*setExpr*⟩ | 'some' ⟨*setExpr*⟩ | 'no' ⟨*setExpr*⟩

⟨*setQuantComplex*⟩ ::= 'all' 'disj' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'all' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'one' 'disj' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'one' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'some' 'disj' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'some' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'no' 'disj' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩
| 'no' ⟨*localDecl*⟩+ '|' ⟨*boolExpr*⟩

⟨*localDecl*⟩ ::= ⟨*localIdentifier*⟩ (';' ⟨*localIdentifier*⟩)* ':' ⟨*setExpr*⟩

⟨*numComp*⟩ ::= ⟨*numExpr*⟩ ≤ ⟨*numExpr*⟩
| ⟨*numExpr*⟩ ≥ ⟨*numExpr*⟩
| ⟨*numExpr*⟩ '=' ⟨*numExpr*⟩
| ⟨*numExpr*⟩ '!=' ⟨*numExpr*⟩

⟨*setExpr*⟩ ::= ⟨*pathExpr*⟩
| ⟨*setExpr*⟩ '++' ⟨*setExpr*⟩
| ⟨*setExpr*⟩ '--' ⟨*setExpr*⟩
| ⟨*setExpr*⟩ '**' ⟨*setExpr*⟩

⟨*numExpr*⟩ ::= '#' ⟨*setExpr*⟩ | '-' ⟨*numExpr*⟩
| ⟨*numExpr*⟩ '+' ⟨*numExpr*⟩
| ⟨*numExpr*⟩ '-' ⟨*numExpr*⟩
| ⟨*numeral*⟩ '*' ⟨*pathExpr*⟩
| ⟨*pathExpr*⟩
| ⟨*numeral*⟩

⟨*pathExpr*⟩ ::= ⟨*identifier*⟩ ('.' ⟨*identifier*⟩)*

⟨*identifier*⟩ ::= 'this' | 'parent' |'dref' | ID

Abbildung 4.1: Grammatik der Expression-Sprache für Referenzen und Constraints in Clafer-Spezifikationen in erweiterter Backus-Naur-Form (EBNF) [75]

- $\twoheadrightarrow \subseteq C \times \langle setExpr \rangle$ ist die *Referenzrelation*,
- $\langle boolExpr \rangle$ ist die Menge der *booleschen Ausdrücke* gemäß der Grammatik in Abbildung 4.1,
- $\Phi \subseteq C \times \langle boolExpr \rangle$ ist die *Constraint-Schachtelungsrelation*.

4.2 WOHLGEFORMTHEITSEIGENSCHAFTEN

Zur Definition der Wohlgeformtheitseigenschaften, die für eine syntaktisch korrekte Clafer-Spezifikation erfüllt sein müssen, führen wir einen Abhängigkeitsgraphen als äquivalente Repräsentation der Schachtelungs-, Vererbungs- und Referenzrelationen zwischen Clafer-Definitionen ein. Im Folgenden verwenden wir, wo es dem Verständnis dienlich ist, zur Notation entweder die Relationen der abstrakten Syntax von Clafer-Definitionen aus Definition 4.8 oder die äquivalenten Kanten des Abhängigkeitsgraphen aus Definition 4.10. Der Abhängigkeitsgraph einer Clafer-Spezifikation fungiert zudem als Basis-Repräsentation für die Transformation der Vererbungsrelationen, für die Transformation von Pfadausdrücken sowie für die Charakterisierung der durch unser Verfahren vollständig analysierbaren Kernsprache.

Die (inversen) Schachtelungs-, (inversen) Vererbungs- und Referenzrelationen einer Clafer-Spezifikation sind als Kanten im Abhängigkeitsgraph repräsentiert. Eine Abhängigkeitsgraph-Kante ist wie folgt definiert:

Definition 4.9 (Kanten im Abhängigkeitsgraph) Eine gerichtete Kante $e \in E$ im Abhängigkeitsgraph ist ein Tupel $e = (c, t, c')$. Die Attribute des Tupels sind wie folgt definiert:

- $c, c' \in C$ sind Quell- und Zielknoten der Kante e,
- $t \in \{ a, s, n, p, r \}$ ist der Typ der Kante e, wobei a eine Vererbungsrelation, s eine inverse Vererbungsrelation, n eine Schachtelungsrelation, p eine inverse Schachtelungsrelation und r eine Referenz repräsentiert,
- $c \xrightarrow{t}{}^{+} c'$ bezeichnet die transitiven Kantenrelationen vom Typ t. Die Clafer-Definitionen c und c' sind in der Relation, falls ausgehend von Clafer-Definition c die Clafer-Definition c' durch Traversierung von Kanten vom Typ t erreichbar ist.

Definition 4.10 (Abhängigkeitsgraph) Der Abhängigkeitsgraph einer Clafer-Spezifikation ist definiert als Tupel $DG = (C, E)$ mit der Knotenmenge C und der Kantenmenge E.

Wir schreiben kurz $c \xrightarrow{t} c'$, um uns auf eine Kante von Quellknoten c zu Zielknoten c' mit Kantentyp t zu beziehen. Eine Vererbungsrelation $c' \twoheadleftarrow c$ entspricht im Abhängigkeitsgraphen einer Kante $c' \xrightarrow{a} c$ und einer Kante $c \xrightarrow{s} c'$ (der inversen Vererbungsbeziehung). Eine Schachtelungsrelation $c' \blacklozenge\!\!\rightarrow c$ entspricht im Abhängigkeitsgraphen einer Kante $c' \xrightarrow{n} c$. Für jede Schachtelungsrelation $c' \blacklozenge\!\!\rightarrow c$ existiert zudem im Abhängigkeitsgraphen die Kante $c \xrightarrow{p} c'$, die eine (inverse) Kind-Eltern-Abhängigkeit der Schachtelungshierarchie repräsentiert.

Für die Charakterisierung von Referenzkanten führen wir folgende partielle Hilfsfunktion ein, die Mengenausdrücke auf Mengen von Clafer-Definitionen abbildet. $\mathcal{P}(C)$ ist hierbei die Potenzmenge von C:

$$\psi : C \rightharpoonup \mathcal{P}(C).$$

Sei `sExp` $\in \langle setExpr \rangle$ der Mengenausdruck, der von der Clafer-Definition c referenziert wird mit $c \twoheadrightarrow$ `sExp`. Die Hilfsfunktion $\psi(c)$ liefert die Menge der Clafer-Definitionen, die in der referenzierten Instanzmenge enthalten sein können. Wenn nun c und `sExp` Teil der Referenz-Relation $c \twoheadrightarrow$ `sExp` sind mit $c \in C$ und `sExp` $\in \langle setExpr \rangle$, dann existieren im Abhängigkeitsgraphen die Kanten $c \xrightarrow{r} c'$ für alle $c' \in \psi(c)$. Die präzise Definition der Hilfsfunktion ψ erfolgt im nächsten Abschnitt mithilfe der statischen Auswertungssemantik von Mengen- und Pfadausdrücken.
Folgendes Beispiel illustriert den Abhängigkeitsgraphen einer Clafer-Spezifikation.

Beispiel 4.11 (Abhängigkeitsgraph einer Clafer-Spezifikation) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Ausschnitt:

```
abstract Node 0..*
  ...
abstract MobileNode : Node 0..*
  memberOf -> DisseminationGroup 0..1
AutonomousNode : MobileNode 0..*
MasterNode : MobileNode 0..*
DisseminationGroup 0..*
  members -> MobileNode -- AutonomousNode
```

Für den gezeigten Ausschnitt ergibt sich die Knotenmenge des Abhängigkeitsgraphen zu $V = \{$ `Node`, `MobileNode`, `AutonomousNode`, `MasterNode`, `DisseminationGroup`, `members` $\}$. Abbildung 4.2 zeigt die Kanten des Abhängigkeitsgraphen. Hierbei sind die Clafer-Definitionen, die sich auf der obersten Ebene der Schachtelungshierarchie befinden, über die Kanten $c_r \xrightarrow{n} c$ und $c \xrightarrow{p} c_r$ mit $c \in \{$ `Node`, `MobileNode`, `AutonomousNode`, `MasterNode`, `DisseminationGroup` $\}$ verbunden. Die Clafer-Definition `members` ist in der Clafer-Definition `DisseminationGroup` geschachtelt und entsprechend durch die Kanten `DisseminationGroup` $\xrightarrow{n}$ `members` und `members` $\xrightarrow{p}$ `DisseminationGroup` verbunden. Die Clafer-Definition `memberOf` ist geschachtelt in der Clafer-Definition `MobileNode`. Entsprechend existieren im Abhän-

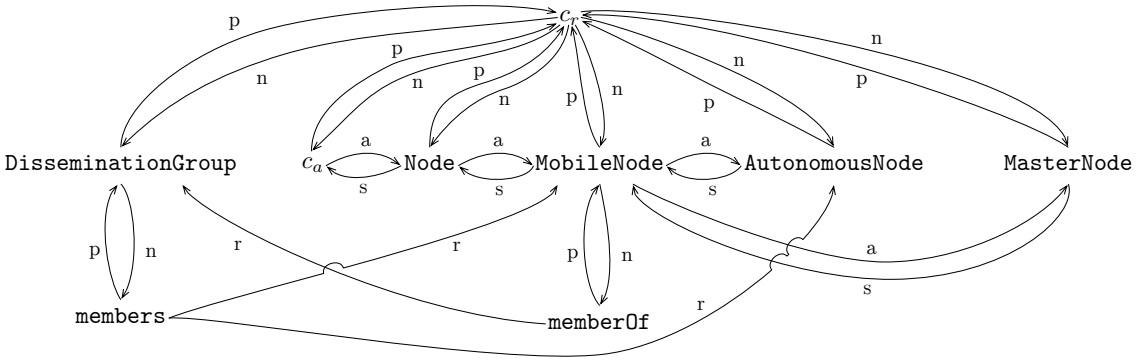


Abbildung 4.2: Beispiel eines Abhängigkeitsgraphen

gigkeitsgraphen die Kanten `MobileNode` $\xrightarrow{n}$ `memberOf` und `memberOf` $\xrightarrow{p}$ `MobileNode`. Die Referenz `members` $\twoheadrightarrow$ `sExp` mit `sExp` $\in \langle setExpr \rangle$ und $\{$ `MobileNode`, `AutonomousNode` $\} = \psi($`members`$)$ ist durch die Kanten `members` $\xrightarrow{r}$ `MobileNode` und `members` $\xrightarrow{r}$ `AutonomousNode` im Abhängigkeitsgraphen repräsentiert. Da die Clafer-Definition `Node` abstrakt ist und sich auf der obersten Ebene der Vererbungshierarchie befindet, existieren im Abhängigkeitsgraphen die Kanten $c_a \xrightarrow{a}$ `Node` und `Node` $\xrightarrow{s} c_a$. Entsprechend ist die Vererbungsbeziehung zwischen Clafer-Definitionen `Node` und `MobileNode` sowie zwischen `MobileNode` und `AutonomousNode` im Abhängigkeitsgraphen durch die Kantenpaar `Node` $\xrightarrow{a}$ `MobileNode`, `MobileNode` $\xrightarrow{s}$ `Node` und `MobileNode` $\xrightarrow{a}$ `AutonomousNode`, `AutonomousNode` $\xrightarrow{s}$ `MobileNode` repräsentiert. Die Repräsentation der Vererbungsrelation zwischen `MobileNode` und `MasterNode` erfolgt entsprechend.

4.2.1 *Wohlgeformtheit von Pfadausdrücken*

Ähnlich wie bei der OCL [33] besteht in der Sprache Clafer ein Pfadausdruck aus einer Sequenz von Bezeichnern id $\in \langle identifier \rangle$, die durch Punkte syntaktisch voneinander getrennt sind. Als Bezeichner in einem Pfadausdruck dürfen Namen id(c) von Clafer-Definitionen $c \in C$ sowie die Schlüsselwörter **`this`**, **`parent`** und **`dref`** auftreten. Wenn ein Pfadausdruck innerhalb eines Mengenausdrucks `sExp` $\in \langle setExpr \rangle$ vorkommt, der von einer Clafer-Definition c referenziert wird (also: $c \twoheadrightarrow$ `sExp`), dann dürfen im Pfadausdruck nur Bezeichner von Clafer-Definitionen und weder **`this`** noch **`parent`** oder **`dref`** auftreten. Einen Pfad, der durch einen Pfadausdruck repräsentiert wird, definieren wir wie folgt:

Definition 4.12 (Pfad) Ein Pfad ist ein n-Tupel $p = (id_1, id_2, \ldots, id_n)$ mit Attributen $id_1, id_2, \ldots, id_n \in \langle identifier \rangle$, wobei n die Länge des Pfades ist. Ein Pfad wird repräsentiert durch einen Pfadausdruck `pExpr` $\in \langle pathExpr \rangle$.

Wir schreiben $p(n)$, um uns auf das n-te Element eines Pfades p zu beziehen.

Ein Pfad lässt sich als Navigation durch den Abhängigkeitsgraphen einer Clafer-Spezifikation interpretieren. Das Schlüsselwort `this` zu Beginn eines Pfadausdrucks erlaubt die Navigation beginnend von einer Instanz der Kontext-Clafer-Definition c_{ctx}. Falls das Schlüsselwort `this` auftritt, ist die Auswertung des Pfadausdrucks abhängig von der Instanz der Kontext-Clafer-Definition. Dies entspricht einer lokalen Interpretation [41]. Falls kein Schlüsselwort `this` zu Beginn eines Pfadausdrucks auftritt, werden Pfadausdrücke vom Wurzelknoten c_r beginnend (global) ausgewertet. Wir definieren die Kontextabhängigkeit von Pfaden wie folgt:

Definition 4.13 (Kontextabhängigkeit von Pfaden und Constraints) Ein Pfad p ist genau dann *kontextabhängig*, falls er mit dem Schlüsselwort `this` beginnt und somit $p(1) =$ `this` gilt. Ansonsten ist ein Pfad *kontextunabhängig*. Ein Constraint ist kontextabhängig, falls er mindestens einen kontextabhängigen Pfad enthält.

Folgendes Beispiel zeigt einen Constraint mit kontextunabhängigen Pfadausdrücken.

Beispiel 4.14 (Kontextunabhängige Pfadausdrücke und Constraints) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Constraint (siehe 31):

```
[ some ProbabilisticBroadcast && some Reliable]
```

Der Constraint ist geschachtelt in der Kontext-Clafer-Definition `Normal` und enthält die beiden Pfadausdrücke `ProbabilisticBroadcast` und `Reliable`. Beide Pfadausdrücke sind kontextunabhängig, da ihr erstes Element nicht dem Schlüsselwort `this` entspricht. Sie können somit unabhängig von der Kontext-Clafer-Definition ausgewertet werden. Der Constraint ist kontextunabhängig, da er keine kontextabhängigen Pfadausdrücke enthält.

Entsprechend zeigt folgendes Beispiel einen Constraint mit kontextabhängigen Pfadausdrücken.

Beispiel 4.15 (Kontextabhängige Pfadausdrücke und Constraints) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Ausschnitt:

```
abstract MobileNode : Node *
  memberOf -> DisseminationGroup ?
    [ this.parent in this.dref.members.dref ]
```

Der gezeigte Constraint ist geschachtelt in der Kontext-Clafer-Definition `memberOf` und enthält die Pfadausdrücke `this.parent` und `this.dref.members.dref`. Beide Pfadausdrücke sind kontextabhängig, da sie mit dem Schlüsselwort `this` beginnen.

> Sie müssen somit ausgehend von der Clafer-Definition `memberOf` ausgewertet werden. Dadurch ist auch der ganze Constraint kontextabhängig.

Zur Definition der statischen Auswertungssemantik von Mengen- und Pfadausdrücken führen wir folgende Relationen ein:

$$s[\![\cdot]\!], s_{ref}[\![\cdot]\!], s_r[\![\cdot]\!] \subseteq \langle setExpr \rangle \times C \times C$$

Damit ein Mengenausdruck $\texttt{sExp} \in \langle setExpr \rangle$ wohlgeformt ist, muss ein Paar $c, c' \in C$ von Clafer-Definitionen existieren, sodass $s[\![\texttt{sExp}]\!](c, c')$ gilt. Damit ein von der Referenz-Clafer-Definition c referenzierter Mengenausdruck $\texttt{sExp} \in \langle setExpr \rangle$ wohlgeformt ist (d. h.: $c \twoheadrightarrow \texttt{sExp}$), muss es eine Clafer-Definition $c' \in C$ geben, sodass $s_{ref}[\![\texttt{sExp}]\!](c, c')$ gilt.

Wegen des Zusammenhangs

$$\texttt{pExpr} \in \langle pathExpr \rangle \implies \texttt{pExpr} \in \langle setExpr \rangle \vee \texttt{pExpr} \in \langle numExpr \rangle$$

aus der Syntax-Definition in Abbildung 4.1, definieren die genannten Relationen insbesondere auch die Wohlgeformtheit von Pfadausdrücken `pExpr`. Die Relationen sind über folgende Gleichungen definiert:

$$s_{ref}[\![\texttt{sExp}]\!](c, c') :\iff c \twoheadrightarrow \texttt{sExp} \wedge \exists c'' \in C : s[\![\texttt{sExp}]\!](c'', c') \quad \boxed{1}$$

$$s[\![\texttt{sExp} \circ \texttt{sExp}']\!](c, c') :\iff s[\![\texttt{sExp}]\!](c, c') \vee s[\![\texttt{sExp}']\!](c, c') \text{ mit } \circ \in \{\texttt{++}, \texttt{--}, \texttt{**}\} \quad \boxed{2}$$

$$s[\![\texttt{e.p}]\!](c, c') :\iff \exists c'' \in C : s[\![e]\!](c, c'') \wedge s_r[\![p]\!](c'', c') \quad \boxed{3}$$

$$s[\![\texttt{this}]\!](c, c') :\iff c = c' = c_{ctx} \quad \boxed{4}$$

$$s[\![\texttt{id}]\!](c, c') :\iff \mathrm{id}(c') = id \wedge c_r \blacklozenge\!\!\rightarrow c' \quad \boxed{5}$$

$$s_r[\![\texttt{e.p}]\!](c, c') :\iff \exists c'' \in C : s_r[\![e]\!](c, c'') \wedge s_r[\![p]\!](c'', c') \quad \boxed{6}$$

$$s_r[\![\texttt{parent}]\!](c, c') :\iff c' \blacklozenge\!\!\rightarrow c \quad \boxed{7}$$

$$s_r[\![\mathrm{id}]\!](c, c') :\iff \underbrace{\left(\mathrm{id}(c') = id \wedge c \blacklozenge\!\!\rightarrow c'\right)}_{\boxed{8.1}} \vee \Big(\underbrace{\neg\exists c'' \in C : \mathrm{id}(c'') = id \wedge c \blacklozenge\!\!\rightarrow c''}_{\boxed{8.2.1}} \wedge \underbrace{\exists c_s \in C_A : c_s \leftarrow c \wedge s_r[\![id]\!](c_s, c')}_{\boxed{8.2.2}}\Big)$$

$$s_r[\![\texttt{dref}]\!](c, c') :\iff \underbrace{\left(\exists \texttt{sExp} \in \langle setExpr \rangle : s_{ref}[\![\texttt{sExp}]\!](c, c')\right)}_{\boxed{9.1}} \vee \Big(\underbrace{\neg\exists \texttt{sExp} \in \langle setExpr \rangle : c \twoheadrightarrow \texttt{sExp}}_{\boxed{9.2.1}} \wedge \underbrace{\exists c_s \in C : c_s \leftarrow^{+} c \wedge s_r[\![\texttt{dref}]\!](c_s, c')}_{\boxed{9.2.2}}\Big)$$

Die Auswertung eines Mengenausdrucks $\texttt{sExp} \in \langle setExpr \rangle$, der von der Clafer-Definition $c \in C$ referenziert wird (d. h. $c \twoheadrightarrow \texttt{sExp}$), ist definiert durch die Relation $s_{ref}[\![\cdot]\!]$ (siehe Gleichung 1). Ein Paar von Clafer-Definitionen c, c' und ein Mengenausdruck sExp ist Teil der Relation genau dann, wenn die Referenzrelation $c \twoheadrightarrow \texttt{sExp}$ besteht und wenn es mindestens eine Clafer-Definition $c'' \in C$ gibt, die Teil der Relation $s[\![\texttt{sExp}]\!](c'', c')$ ist. Im Abhängigkeitsgraphen existiert genau dann eine Referenzkante $c \xrightarrow{r} c'$, wenn die Relation $s_{ref}[\![\texttt{sExp}]\!](c, c')$ besteht. Die Hilfsfunktion ψ lässt sich somit definieren als:

$$\psi(c) := \{\, c' \in C \mid s_{ref}[\![\texttt{sExp}]\!](c, c') \,\}.$$

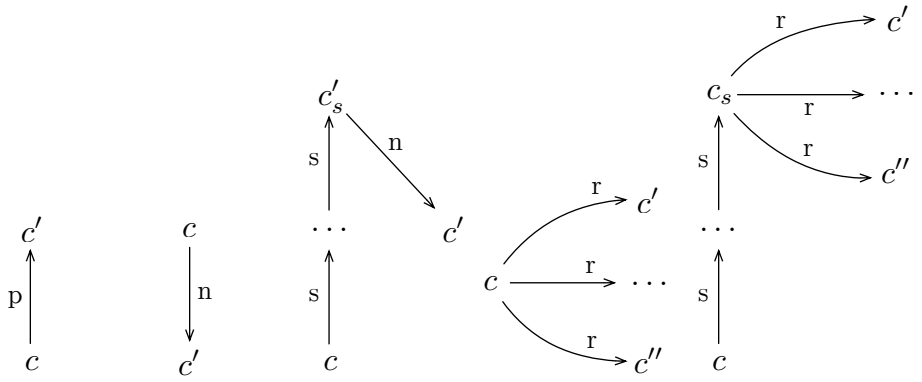
Die Auswertung eines binären Mengenausdrucks $\texttt{sExp} \circ \texttt{sExp}'$ mit $\circ \in \{\texttt{++}, \texttt{--}, \texttt{**}\}$ für ein Paar von Clafer-Definitionen $c, c' \in C$ ist induktiv durch die Relation $s[\![\cdot]\!]$ definiert (siehe Gleichung 2). Falls das Paar von Clafer-Definitionen c, c' Teil der Relation $s[\![\texttt{sExp} \circ \texttt{sExp}']\!](c, c')$ ist, müssen sie auch Teil der Relationen $s[\![\texttt{sExp}]\!](c, c')$ und $s[\![\texttt{sExp}']\!](c, c')$ sein.

Die Auswertung eines Pfadausdrucks $s[\![\texttt{e.p}]\!](c, c')$ für einen gegebenen Kontext c_{ctx} ist induktiv durch Gleichung 3 definiert, welche die Auswertung von Teilpfaden e und p für Clafer-Definitionen c und c' transitiv zueinander in Beziehung setzt. Es muss demnach mindestens eine Clafer-Definition c'' existieren, die Teil der Relationen $s[\![\texttt{e}]\!](c, c'')$ und $s_r[\![\texttt{p}]\!](c'', c')$ ist. Falls die Auswertung für einen kontextunabhängigen Pfadausdruck erfolgt (insbesondere bei der Auswertung von referenzierten Mengenausdrücken), entspricht die Kontext-Clafer-Definition der Wurzel-Clafer-Definition (d. h. mit $c_{ctx} = c_r$).

Die Relation $s[\![\cdot]\!]$ ist definiert für den Teilpfad e, der dem ersten Element eines Pfades entspricht. Die Relation $s_r[\![\cdot]\!]$ ist induktiv für die Auswertung der verbliebenen Elemente des Teilpfades p durch Gleichung 6 definiert. Das erste Element eines Pfades kann entweder einem Teilpfad mit dem Schlüsselwort **this** oder einem (globalen) Bezeichner $\mathrm{id}(c) = id$ einer Clafer-Definition c entsprechen. Für die Relation zur Auswertung des Schlüsselwortes **this** gilt, dass c und c' der Kontext-Clafer-Definition c_{ctx} entsprechen müssen (siehe Gleichung 4). Hierdurch ist sichergestellt, dass die Auswertung eines kontextabhängigen Pfadausdrucks von der Kontext-Clafer-Definition c_{ctx} ausgeht. Für die Relation $s[\![\texttt{id}]\!](c, c')$ zur Auswertung eines globalen Bezeichners (mit $p(1) = \mathrm{id}$), muss ausgehend von der Wurzel-Clafer-Definition c_r eine Clafer-Definition c' existieren, für welche die Bedingung $\mathrm{id}(c') = id \wedge c_r \blacklozenge\!\!\rightarrow c'$ gilt (siehe Gleichung 5).

Durch das Pfadelement $p(i) =$ **parent** lässt sich ausgehend von einer Kontext-Clafer-Definition c in der Schachtelungshierarchie aufwärts navigieren. Im Abhängigkeitsgraphen entspricht dies dem Traversieren der Kante $c_{ctx} \xrightarrow{p} c$ (siehe Abbildung 4.3a). Für die Auswertung eines Teilpfades **parent** muss das Paar von Clafer-Definitionen $c, c' \in C$ somit sowohl Teil der Relation $s_r[\![\textbf{parent}]\!](c, c')$ als auch Teil der Schachtelungsrelation $c' \blacklozenge\!\!\rightarrow c$ sein (siehe Gleichung 3).

Durch den Teilpfad $p(i) = \mathrm{id}$ mit $\mathrm{id}(c') = \mathrm{id}$ lässt sich ausgehend von einer Clafer-Definition c in der Schachtelungs- bzw. Vererbungshierarchie abwärts zu einer Clafer-Definition c' navigieren. Im Abhängigkeitsgraphen entspricht dies dem Traversieren der



(a) Aufwärts **(b)** Abwärts **(c)** Abwärts & Vererbung **(d)** Referenz **(e)** Referenz & Vererbung

Abbildung 4.3: Traversieren von Pfaden im Abhängigkeitsgraph

Kante $c \xrightarrow{n} c'$ (siehe Abbildung 4.3b). Falls eine in der Vererbungshierarchie übergeordnete Clafer-Definition c_s mit $c_s \leftarrow^{+} c$ existiert und die Clafer-Definition c' in c_s geschachtelt ist, entspricht dies dem Traversieren der Kanten $c \xrightarrow{s} \ldots \xrightarrow{s} c_s \xrightarrow{n} c'$ (siehe Abbildung 4.3c). Die Auswertung eines Teilpfades `id`, der nicht am Beginn eines Pfadausdrucks auftritt, lässt sich somit durch die Relation $s_r[\![\text{id}]\!](c, c')$ definieren, die für ein Paar der Clafer-Definitionen $c, c' \in C$ immer genau dann besteht, falls mindestens eine der folgenden Bedingungen gilt: (i) Mindestens eine Clafer-Definition c' ist in der Clafer-Definition c geschachtelt (siehe Bedingung 8.1). (ii) Falls keine entsprechend geschachtelte Clafer-Definition existiert (siehe Bedingung 8.2.1), muss mindestens eine Clafer-Definition c' existieren, die in einer zur Clafer-Definition c in der Vererbungshierarchie übergeordneten Clafer-Definition c_s geschachtelt ist, und für welche die Bedingung $s_r[\![id]\!](c_s, c')$ gilt (siehe Bedingung 8.2.2).

Durch das Schlüsselwort **`dref`** lässt sich ausgehend von der Clafer-Definition c über Referenzen $c \twoheadrightarrow$ `sExp` hinweg navigieren. Im Abhängigkeitsgraphen entspricht dies dem Traversieren der Kante $c \xrightarrow{r} c'$ mit $c' \in \psi(c)$ (siehe Abbildung 4.3d). Da in einer Referenzrelation $c \twoheadrightarrow$ `sExp` der Mengenausdruck `sExp` potentiell mehr als eine Clafer-Definition berechnet, existiert im Abhängigkeitsgraphen für jede Clafer-Definition $c' \in \psi(c)$ jeweils eine Kante $c \xrightarrow{r} c'$. Die Auswertung eines Teilpfades **`dref`** lässt sich somit für ein Paar von Clafer-Definitionen $c, c' \in C$ durch die Relation $s_r[\![\textbf{dref}]\!](c, c')$ definieren, die genau dann besteht, falls die Clafer-Definition c mindestens einen Mengenausdruck `sExp` referenziert der Teil der Relation $s_{ref}[\![\texttt{sExp}]\!](c, c')$ ist (siehe Gleichung 9.1).

Falls keine Referenz $c \twoheadrightarrow$ `sExp` existiert (siehe Gleichung 9.2.1), dann lässt sich durch das Schlüsselwort **`dref`** ausgehend von einer Clafer-Definition c über eine Referenz $c_s \twoheadrightarrow \texttt{sExp}_s$ navigieren, wobei Clafer-Definition c_s in der Vererbungshierarchie c übergeordnet ist (d. h. $c_s \leftarrow^{+} c$). Im Abhängigkeitsgraphen entspricht dies dem Traversieren der Kanten

$c \xrightarrow{s} \ldots \xrightarrow{s} c_s \xrightarrow{n} c'$ mit $c' \in \texttt{sExp}_s$ (siehe Abbildung 4.3e). Auch in diesem Fall kann der Pfadausdruck zu mehreren Pfaden im Abhängigkeitsgraphen aufgelöst werden, falls der Mengenausdruck $\texttt{sExp}_s$ mehr als eine Clafer-Definition berechnet. Ein Paar aus Clafer-Definitionen c und c' ist in diesem Fall genau dann Teil der Relation $s_r[\![\texttt{dref}]\!](c, c')$, wenn die Clafer-Definition c (transitiv) von einer Clafer-Definition c_s erbt und Teil der Relation $s[\![\texttt{dref}]\!](c_s, c')$ ist (siehe Bedingung 9.2.2).

Folgendes Beispiel demonstriert die Überprüfung der Wohlgeformtheit von Pfadausdrücken.

Beispiel 4.16 (Wohlgeformtheit von Pfadausdrücken) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Ausschnitt:

```
abstract Node *
...
abstract MobileNode : Node *
  memberOf -> DisseminationGroup ?
AutonomousNode : MobileNode *
MasterNode : MobileNode *
DisseminationGroup *
  members -> MobileNode -- AutonomousNode
    [ this.parent in this.dref.memberOf.dref ] 38
```

Der Constraint 38 ist in der Clafer-Definition `members` geschachtelt und enthält die Pfadausdrücke `this.parent` sowie `this.dref.memberOf.dref`. Da beide Pfadausdrücke kontextabhängig sind, beginnt die Auswertung ausgehend von der Kontext-Clafer-Definition `members`. Es gilt somit die Bedingung $c_{ctx} = \texttt{members}$. Abbildung 4.4 zeigt den Abhängigkeitsgraphen des betrachteten Ausschnitts der Spezifikation. Wir untersuchen als Nächstes die Wohlgeformtheit der beiden Pfadausdrücke, indem wir die Relationen $s[\![\cdot]\!], s_r[\![\cdot]\!]$ in Bezug auf die Teilpfade untersuchen, die durch die Gleichungen 1 bis 9 zuvor definiert wurden. Die Untersuchung der Wohlgeformtheit des Pfadausdrucks `this.parent` ergibt sich wie folgt:

- Anwendung auf Pfad (mit Gleichung 3):

  $s_r[\![\texttt{this.parent}]\!](\texttt{members}, \texttt{DisseminationGroup}) : \Longleftrightarrow$

  $\quad s[\![\texttt{this}]\!](\texttt{members}, \texttt{members}) \wedge s_r[\![\texttt{parent}]\!](\texttt{members}, \texttt{DisseminationGroup})$

- Anwendung auf Teilpfad (mit Gleichung 4):

  $s[\![\texttt{this}]\!](\texttt{members}, \texttt{members}) : \Longleftrightarrow \ \texttt{members} = c_{ctx}$

- Anwendung auf Teilpfad (mit Gleichung 7):

  $s_r[\![\texttt{parent}]\!](\texttt{members}, \texttt{DisseminationGroup}) : \Longleftrightarrow \ \texttt{DisseminationGroup} \blacklozenge\!\!\rightarrow \texttt{members}$

Der Pfadausdruck kann ausgehend von der Clafer-Definition `members` somit zu der Clafer-Definition `DisseminationGroup` aufgelöst werden. Dies entspricht dem Traversieren der Kante $\texttt{members} \xrightarrow{p} \texttt{DisseminationGroup}$ im Abhängigkeitsgraph.

Die Untersuchung der Wohlgeformtheit des Pfadausdrucks **this**.**dref**.memberOf.**dref** ergibt sich wie folgt:

- Anwendung auf den Pfad **this**.**dref**.memberOf.**dref** (mit Gleichung 3):

  $s[\![\textbf{this}.\textbf{dref}.\text{memberOf}.\textbf{dref}]\!](\text{members},\text{DisGroup}) : \Longleftrightarrow$

  $s[\![\textbf{this}]\!](\text{members},\text{members}) \wedge s_r[\![\textbf{dref}.\text{memberOf}.\textbf{dref}]\!](\text{members},\text{DisGroup})$

- Für den Teilpfad **dref**.memberOf.**dref** ergeben sich zwei Belegungen (mit Gleichung 6):

  $s_r[\![\textbf{dref}.\text{memberOf}.\textbf{dref}]\!](\text{members},\text{memberOf}) : \Longleftrightarrow$

  $s_r[\![\textbf{dref}]\!](\text{members},\text{MobileNode}) \wedge s_r[\![\text{memberOf}.\textbf{dref}]\!](\text{MobileNode},\text{DisGroup}) \vee$

  $s[\![\textbf{dref}]\!](\text{members},\text{AutonomousNode}) \wedge s_r[\![\text{memberOf}.\textbf{dref}]\!](\text{AutonomousNode},\text{DisGroup})$

- Entsprechend ergeben sich für den Teilpfad memberOf.**dref** zwei Belegungen:

  $s[\![\text{memberOf}.\textbf{dref}]\!](\text{MobileNode},\text{DisGroup}) : \Longleftrightarrow$

  $s[\![\text{memberOf}]\!](\text{MobileNode},\text{memberOf}) \wedge s_r[\![\textbf{dref}]\!](\text{memberOf},\text{DisGroup})$

  $s[\![\text{memberOf}.\textbf{dref}]\!](\text{AutonomousNode},\text{DisGroup}) : \Longleftrightarrow$

  $s[\![\textbf{this}]\!](\text{AutonomousNode},\text{memberOf}) \wedge s_r[\![\textbf{dref}]\!](\text{memberOf},\text{DisGroup})$

- Für Teilpfad **this** ergibt sich mit Gleichung 4 folgende Belegung:

  $s[\![\textbf{this}]\!](\text{members},\text{members}) : \Longleftrightarrow \ \text{members} = c_{ctx}$

- Für die Teilpfade **dref** ergeben sich (mit Gleichung 9.1) drei Belegungen:

  $s_r[\![\textbf{dref}]\!](\text{members},\text{MobileNode}) : \Longleftrightarrow \ \text{members} \twoheadrightarrow \underbrace{\text{MobileNode} -\!- \text{AutonomousNode}}_{\text{sExp}} \wedge$

  $\text{MobileNode} \in \psi(\text{members}) = \{\text{MobileNode},\text{AutonomousNode}\}$

  $s_r[\![\textbf{dref}]\!](\text{members},\text{AutonomousNode}) : \Longleftrightarrow \ \text{members} \twoheadrightarrow \underbrace{\text{MobileNode} -\!- \text{AutonomousNode}}_{\text{sExp}'} \wedge$

  $\text{AutonomousNode} \in \psi(\text{members}) = \{\text{MobileNode},\text{AutonomousNode}\}$

  $s_r[\![\textbf{dref}]\!](\text{memberOf},\text{DisseminationGroup}) : \Longleftrightarrow \ \text{memberOf} \twoheadrightarrow \text{DisseminationGroup} \wedge$

  $\text{DisseminationGroup} \in \psi(\text{DisseminationGroup}) = \{\text{DisseminationGroup}\}$

- Für den Teilpfad memberOf ergeben sich (mit Gleichung 8.1) zwei Belegungen:

  $s_r[\![\text{memberOf}]\!](\text{MobileNode},\text{memberOf}) : \Longleftrightarrow \ \text{MobileNode} \blacklozenge\!\!\rightarrow \text{memberOf}$

  $s_r[\![\text{memberOf}]\!](\text{AutonomousNode},\text{memberOf}) : \Longleftrightarrow$

  $\text{MobileNode} \leftarrow \text{AutonomousNode} \wedge s_r[\![\text{memberOf}]\!](\text{MobileNode},\text{memberOf})$

Die Auswertung hat ergeben, dass der Pfadausdruck wohlgeformt ist und zu der Clafer-Definition DisseminationGroup aufgelöst werden kann. Im Abhängigkeitsgraphen ergeben sich zwei Kantenfolgen, die beim Auflösen traversiert werden:

(i) $\text{members} \xrightarrow{r} \text{MobileNode} \xrightarrow{n} \text{memberOf} \xrightarrow{r} \text{DisGroup}$,

(ii) $\text{members} \xrightarrow{r} \text{AutonomousNode} \xrightarrow{s} \text{MobileNode} \xrightarrow{n} \text{memberOf} \xrightarrow{r} \text{DisGroup}$.

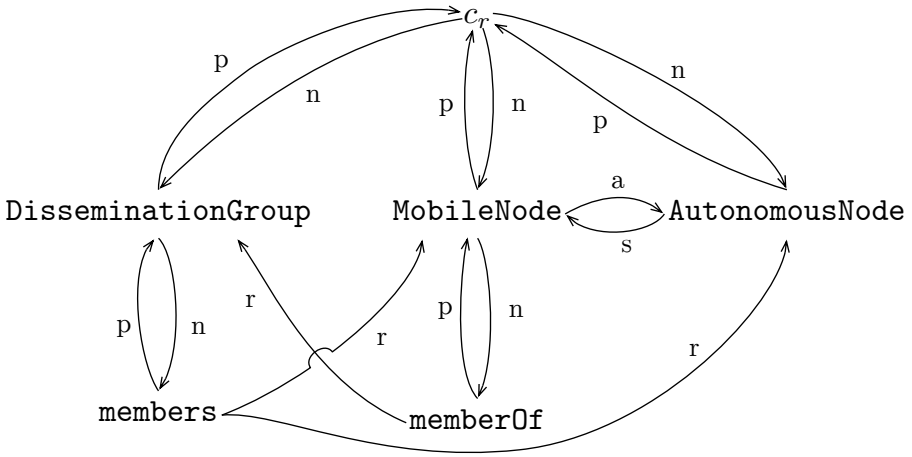


Abbildung 4.4: Auswertung von Pfaden im Abhängigkeitsgraph

4.2.2 *Grundlegende Wohlgeformheitseigenschaften*

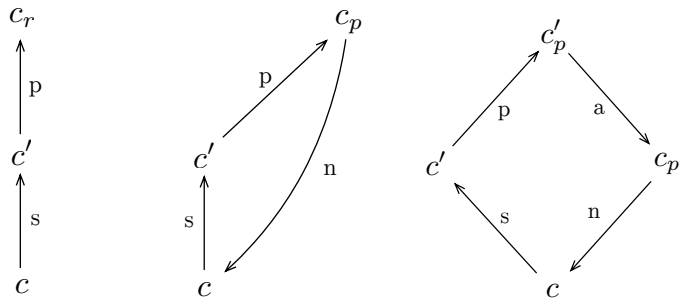
Mithilfe der eingeführten abstrakten Syntax, dem Abhängigkeitsgraphen und der zuvor definierten statischen Auswertungssemantik für Pfadausdrücke definieren wir in diesem Abschnitt grundlegende Wohlgeformtheitseigenschaften. Für eine syntaktisch korrekte Clafer-Spezifikation müssen folgende Wohlgeformtheitseigenschaften erfüllt sein:

- Der Bezeichner einer Clafer-Definition ist eindeutig für alle Clafer-Definitionen, die in der gleichen Clafer-Definition geschachtelt sind. Es muss somit die Bedingung $\forall c, c', c'' : c'' \xrightarrow{n} c \wedge c'' \xrightarrow{n} c' \wedge c \neq c' \implies \mathrm{id}(c) \neq \mathrm{id}(c')$ gelten.
- Die Hilfs-Clafer-Definition c_r muss in einer Clafer-Spezifikationsinstanz genau eine Instanz aufweisen. Somit gilt die Bedingungen $\lambda_c(c_r) = (1,1)$. Darüber hinaus darf die Hilfs-Clafer-Definition c_r über beliebig viele Sub-Clafer-Definitionen verfügen. Somit muss die Bedingung $\lambda_g(c_r) = (0,*)$ erfüllt sein. Zusätzlich muss die Bedingung $c_r \xrightarrow{n} c_a$ gelten.
- Die Schachtelungsrelation $\blacklozenge\!\!\rightarrow$ bildet einen endlichen Wurzelbaum auf der Menge C mit der Wurzel c_r. Jeder Knoten im Abhängigkeitsgraphen (außer der Wurzel-Clafer-Definition c_r) hat daher genau eine ausgehende Kante, die eine inverse Schachtelungsrelation repräsentiert. Es muss somit die Bedingung $\forall c \in C \setminus \{ c_r \} : \exists! c' \in C : c \xrightarrow{p} c'$ gelten.
- Die Vererbungsrelation $\twoheadleftarrow$ bildet einen endlichen Wurzelbaum auf der Menge $C_A \cup C'_O$ mit der Wurzel c_a (mit $C'_O = \{ c \in C_O \mid c' \in C_A \wedge c' \twoheadleftarrow c \}$). Jeder abstrakte Knoten im Abhängigkeitsgraphen (außer c_a) hat somit genau eine ausgehende Kante, die eine inverse Vererbungsrelation repräsentiert. Es muss somit die Bedingung $\forall c \in C_A \setminus \{ c_a \} \cup C'_O : \exists! c' \in C : c \xrightarrow{s} c'$ gelten.

- Für die Verfeinerung von Multiplizitätsintervallen durch Vererbungsrelationen $c' \twoheadleftarrow c$ zwischen Clafer-Definitionen $c, c' \in C$ mit $\lambda_c(c') = (l', u')$ und $\lambda_c(c) = (l, u)$ müssen die Bedingungen $l' \leq l$ und $u \leq u'$ gelten. Falls $u' = *$, darf u beliebig groß gewählt werden. Auf diese Weise wird sichergestellt, dass erbende Clafer-Definitionen eine Spezialisierung ihrer in der Vererbungshierarchie übergeordneten abstrakten Clafer-Definitionen bilden. Für die abstrakte Clafer-Definition $c_a \in C$ gilt die Bedingung $\lambda_c(c_a) = (0, *)$. Für die Verfeinerung von Gruppenkardinalitätsintervallen λ_g existiert die genannte Einschränkung nicht, sodass erbende Clafer-Definitionen ihr Gruppenkardinalitätsintervall beliebig erweitern und so zusätzliche Clafer-Definitionen schachteln können.

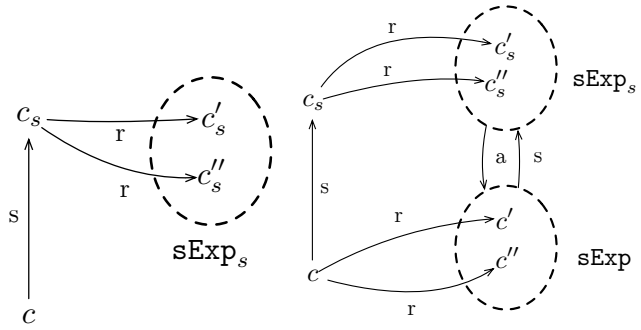
- Vererbungsrelationen $\twoheadleftarrow$ sind abhängig von Schachtelungsrelationen $\blacklozenge\!\!\rightarrow$. Falls eine Vererbungsrelation $c' \twoheadleftarrow c$ existiert, muss genau eine der folgenden Bedingungen erfüllt sein:
  - Clafer-Definition c' ist geschachtelt in Wurzel-Clafer-Definition c_r. Im Abhängigkeitsgraphen müssen somit die Kanten $c \xrightarrow{s} c' \xrightarrow{p} c_r$ existieren. Auf diese Weise wird sichergestellt, dass beliebig geschachtelte Clafer-Definitionen von global definierten abstrakten Clafer-Definitionen erben können. Abbildung 4.5a stellt diesen Zusammenhang im Abhängigkeitsgraphen dar.
  - Clafer-Definition c' ist in derselben Eltern-Clafer-Definition c_p geschachtelt, wie Clafer-Definition c. Im Abhängigkeitsgraphen existiert somit ein Zyklus bestehend aus den Kanten $c \xrightarrow{s} c' \xrightarrow{p} c_p \xrightarrow{n} c$ (siehe Abbildung 4.5b). Auf diese Weise wird sichergestellt, dass Clafer-Definitionen von lokal definierten geschachtelten abstrakten Clafer-Definitionen erben können.
  - Clafer-Definition c' ist in einer abstrakten Clafer-Definition c'_p geschachtelt, von der Clafer-Definition c_p erbt. Clafer-Definition c ist in Clafer-Definition c_p geschachtelt. Im Abhängigkeitsgraphen existiert somit ein Zyklus bestehend aus den Kanten $c \xrightarrow{s} c' \xrightarrow{p} c'_p \xrightarrow{a} c_p \xrightarrow{n} c$ (siehe Abbildung 4.5c). Auf diese Weise wird sichergestellt, dass geschachtelte erbende Clafer-Definitionen konsistent zu den Schachtelungsrelationen ihrer in der Vererbungshierarchie übergeordneten abstrakten Clafer-Definitionen sind.

- Für die Verfeinerung einer Referenzrelation $c_s \twoheadrightarrow \texttt{sExp}_s$ durch eine Vererbungsrelation $c_s \twoheadleftarrow^{+} c$ mit $c \in C$, $c_s \in C_A$ und $\texttt{sExp}_s \in \langle setExpr \rangle$ muss Folgendes gelten:
  - Falls keine Relation $c \twoheadrightarrow \texttt{sExp}$ existiert, besteht keine Einschränkung. In diesem Fall erbt c die Referenzen von c_s. Auf diese Weise wird sichergestellt, dass von einer erbenden Clafer-Definition zu Instanzen navigiert werden kann, die von ihrer in der Vererbungshierarchie übergeordneten Clafer-Definition referenziert werden. Der Abhängigkeitsgraph dieses Falles ist in Abbildung 4.6a dargestellt.



(a) Erster Fall **(b)** Zweiter Fall **(c)** Dritter Fall

Abbildung 4.5: Zusammenhang zwischen Vererbungs- und Schachtelungsrelationen im Abhängigkeitsgraphen



(a) Erster Fall **(b)** Zweiter Fall

Abbildung 4.6: Zusammenhang zwischen Vererbungs- und Referenzrelationen im Abhängigkeitsgraphen

– Falls zusätzlich eine Referenzrelation $c \twoheadrightarrow \texttt{sExp}$ mit $\texttt{sExp} \in \langle \mathit{setExpr} \rangle$ existiert, muss für jede Clafer-Definition $c' \in \psi(c)$ mindestens eine Clafer-Definition $c'_s \in \psi(c_s)$ existieren, von der Clafer-Definition c' erbt. In diesem Fall muss somit die Bedingung $\forall c' \in \psi(c)\, \exists c'_s \in \psi(c_s) : c'_s \xrightarrow{a}{}^{+} c'$ erfüllt sein. Auf diese Weise wird wieder sichergestellt, dass von erbenden Clafer-Definitionen über die Referenzen der in der Vererbungshierarchie übergeordneten Clafer-Definition navigiert werden kann. Es werden Instanzen der Clafer-Definition c' referenziert, die Spezialisierungen der Clafer-Definition c'_s sind. Abbildung 4.6b zeigt den Abhängigkeitsgraph des zweiten Falls.

Auf Basis der Syntax- und Wohlgeformtheitsdefinitionen können wir nun die *syntaktische Unbeschränktheit* von Clafer-Spezifikationen definieren. Eine syntaktisch unbeschränkte Clafer-Spezifikation kann über eine potentiell beliebig große Anzahl von Clafer-Spezifikationsinstanzen verfügen. Eine Clafer-Spezifikation ist genau dann syntaktisch

unbeschränkt, wenn mindestens eine Clafer-Definition syntaktisch unbeschränkt ist. Eine Clafer-Definition ist wiederum genau dann syntaktisch unbeschränkt, wenn sowohl ihr Multiplizitätsintervall als auch das Gruppenkardinalitätsintervall der Eltern-Clafer-Definition unbeschränkt ist. Alternativ ist eine Clafer-Definition syntaktisch unbeschränkt, falls mindestens eine in der Schachtelungshierarchie übergeordnete Clafer-Definition syntaktisch unbeschränkt ist.

Beispiel 4.17 (Syntaktische Unbeschränktheit von Clafer-Definitionen) In der in Listing 3.1 gezeigten Clafer-Spezifikation ist die Clafer-Definition `Node` mit dem Multiplizitätsintervall $0..*$ annotiert. Die Clafer-Definition `Node` ist somit syntaktisch unbeschränkt. Des Weiteren ist die Clafer-Definition `Interface` geschachtelt in der Clafer-Definition `Node` und mit dem Multiplizitätsintervall 1..1 annotiert. Die Clafer-Definition `Interface` ist ebenfalls syntaktisch unbeschränkt, da die in der Schachtelungshierarchie übergeordnete Clafer-Definition `Node` syntaktisch unbeschränkt ist.

Die syntaktische Unbeschränktheit von Clafer-Spezifikationen und Clafer-Definitionen lässt sich somit folgendermaßen definieren:

Definition 4.18 (Syntaktische Unbeschränktheit) Eine Clafer-Definition $c \in C$ ist syntaktisch unbeschränkt, falls die Clafer-Definitionen $c', c'' \in C$ existieren, sodass folgende Bedingungen alle erfüllt sind:

(i) Clafer-Definition c'' ist Eltern-Clafer-Definition von c', d. h. $c'' \blacklozenge\!\!\rightarrow c'$,

(ii) das Gruppenkardinalitätsintervall von c'' ist unbeschränkt, d. h. $(l, *) := \lambda_g(c)$,

(iii) das Multiplizitätsintervall von c' ist unbeschränkt, d. h. $(l, *) = \lambda_c(c)$,

(iv) Clafer-Definition c' entspricht c oder liegt in der Schachtelungshierarchie oberhalb von c, d. h. $c' \blacklozenge\!\!\rightarrow^* c$.

Eine Clafer-Spezifikation ist syntaktisch unbeschränkt, wenn sie mindestens eine Clafer-Definition enthält, die syntaktisch unbeschränkt nach dieser Definition ist.

Eine Clafer-Spezifikation, die syntaktisch unbeschränkt ist, spezifiziert nicht zwangsläufig eine beliebig große Anzahl von Clafer-Spezifikationsinstanzen. Um die tatsächliche Unbeschränktheit, Anomalien und weitere semantische Eigenschaften von Clafer-Spezifikationen zu analysieren, wird daher im Folgenden die semantische Repräsentation der Sprache Clafer eingeführt.

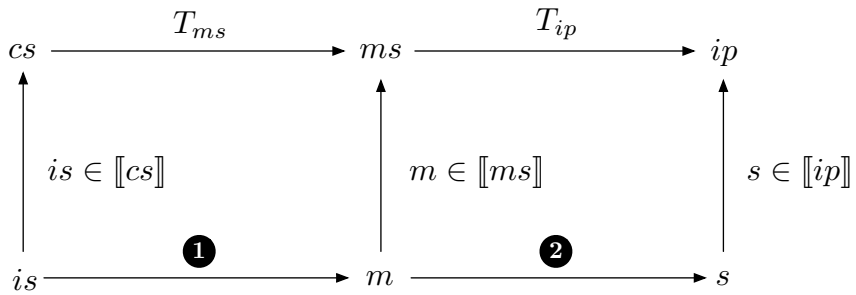


Abbildung 4.7: Überblick über verwendete Sprachen und deren semantische Repräsentation

4.3 SEMANTIK VON CLAFER-SPEZIFIKATIONEN

Zur Analyse einer Clafer-Spezifikation wird diese zunächst in mehrere Zwischenrepräsentationen übersetzt, sodass die Zielrepräsentation letztlich als mathematisches Optimierungsproblem mit Standard-Solvern untersucht werden kann. Abbildung 4.7 zeigt in einem Diagramm die Sprachen der Zwischenrepräsentationen, deren semantische Repräsentationen sowie die Übergänge und deren Notation, die im Folgenden erläutert werden.

4.3.1 *Repräsentation von Clafer-Spezifikationsinstanzen*

Die Menge $[\![cs]\!]$ beinhaltet alle Clafer-Spezifikationsinstanzen einer Clafer-Spezifikation cs und ist potentiell unbeschränkt. Eine einzelne Clafer-Spezifikationsinstanz $is \in [\![cs]\!]$ wird repräsentiert durch eine Menge von Clafer-Definitions-Instanzen $is = \{ i_1, i_2, \dots \}$, wobei jede Clafer-Definitions-Instanz genau einer Clafer-Definition $c \in C$ zugeordnet ist. Umgekehrt können hingegen einer Clafer-Definition mehrere Clafer-Definitions-Instanzen zugeordnet sein. Wir schreiben $is(c)$, um auf die Teilmenge der Clafer-Definitions-Instanzen (von Clafer-Spezifikationsinstanz $is \in [\![cs]\!]$) zu verweisen, die Clafer-Definition c zugeordnet sind. Die Instanzen der Clafer-Definitionen müssen die Bedingungen der Multiplizitätsintervalle erfüllen. Zudem müssen alle Constraints der Spezifikation durch die Clafer-Spezifikationsinstanz erfüllt sein[1].

> **Beispiel 4.19** (Repräsentation von Clafer-Spezifikationsinstanzen) Die in Abschnitt 3.2.1 gezeigte Clafer-Spezifikationsinstanz is' enthält insgesamt 43 Clafer-Definitions-Instanzen. Eine Clafer-Spezifikationsinstanz kann in der zuvor eingeführten Notation als Menge $is' = \{ i_1, i_2, \dots, i_{43} \}$ repräsentiert werden, wobei die Indizes der Menge

1 Für eine ausführliche Charakterisierung der Bedingungen, die für gültige Clafer-Spezifikationsinstanzen gelten müssen, sei an dieser Stelle auf [10] verwiesen.

den Zeilennummern in der Abbildung entsprechen. Die Zuordnung der Clafer-Definitions-Instanzen zu ihren jeweiligen Clafer-Definitionen `members` und `Protocol` ist durch die Notation $is(\texttt{members}) := \{i_{41}, i_{42}, i_{43}\}$ bzw. $is(\texttt{Protocol}) := \{i_{12}, i_{25}, i_{30}\}$ ersichtlich.

Definition 4.20 (Clafer-Spezifikationsinstanz) Eine Clafer-Spezifikationsinstanz ist definiert als Menge von Clafer-Definitions-Instanzen $is = \{i_1, i_2, \ldots\}$ mit $is \in [\![cs]\!]$. Die Menge $[\![cs]\!]$ enthält alle Clafer-Spezifikationsinstanzen, welche die Constraints und Bedingungen der Multiplizitätsintervalle einer Clafer-Spezifikation cs erfüllen.

4.3.2 *Multimengen-Repräsentation*

Um die effiziente Analyse von Clafer-Spezifikationen mit einer potentiell unbeschränkten Anzahl von Clafer-Spezifikationsinstanzen zu ermöglichen, führen wir eine Multimengen-Repräsentation für Clafer-Spezifikationen ein. Die Repräsentation basiert auf mehreren symbolischen Multimengen M_P über Teilmengen des Universums C und zusätzlichen Bedingungen zwischen Elementen der Multimengen. Die *Zählfunktion* einer Multimenge $m_P : C \to V$ ist die Abbildung der Menge C auf eine Menge von Variablensymbolen $V \subseteq \mathbb{V}$. Durch dieses Abbildung wird jeder Clafer-Definition ein Variablensymbol zugeordnet, das einen ganzzahligen Wert repräsentiert. Eine symbolische Multimenge ist ein Tupel $M_P = (P, V, m_P)$ und ermöglicht es, die Anzahl der instanziierten Clafer-Definitionen einer Clafer-Spezifikationsinstanz zu repräsentieren. Die Menge $\mathbb{V}$ bezeichnet die Variablen aller symbolischen Multimengen, die disjunkt gegenüber den Multimengen sind, denen sie zugeordnet sind. Wir schreiben $\mathrm{Supp}(M_P) := P$, um uns auf die *Trägermenge* P (engl. supporting set) einer Multimenge M_P zu beziehen. Für alle Clafer-Definitionen, die nicht Teil der Trägermenge sind, muss die Anzahl der durch die Multimenge repräsentierten Instanzen gleich null sein. Für alle Clafer-Definitionen, die Teil der Trägermenge sind, muss die Anzahl der durch die Multimenge repräsentierten Instanzen größer oder gleich null sein. Es muss somit gelten: $P \subseteq C$. Die Menge aller symbolischen Multimengen des Universums C kennzeichnen wir als $\mathbb{M}_C \subseteq \mathbb{V}^C$.

Beispiel 4.21 (Symbolische Multimengen und Zählfunktionen) Sei M_P eine symbolische Multimenge, die Instanzen der Clafer-Definitionen `DisseminationGroup`, `Strategy`, `Central` und `members` repräsentieren soll, dann ist die Trägermenge $P = \mathrm{Supp}(M_P) = \{\texttt{DisseminationGroup}, \texttt{Strategy}, \texttt{Central}, \texttt{members}\}$. Für alle anderen symbolischen Zählfunktionswerte der Clafer-Definitionen, die nicht zur Trägermenge gehören (d. h. $c \in C \setminus P$), gilt für mögliche Variablenbelegungen die Forderung: $m_P(c) = 0$.

Definition 4.22 (Symbolische Multimenge) Sei C die Menge aller Clafer-Definitionen einer Clafer-Spezifikation $cs \in CS$. Eine symbolische Multimenge ist ein Tupel $M_P = (P, V, m_P)$. Die Attribute des Tupels M_P sind wie folgt definiert:

- $P \subseteq C$ ist die *Trägermenge* der symbolischen Multimenge.
- $V \subseteq \mathbb{V}$ ist eine Menge von *Variablensymbolen*.
- $m_P : C \to V$ ist die *Zählfunktion* der symbolischen Multimenge.

Die Menge $\mathbb{V}$ enthält die Variablensymbole aller symbolischen Multimengen. Die Variablenmengen aller symbolischen Multimengen sind in $\mathbb{V}$ disjunkt. Weiterhin ist $\mathbb{M}_C \subseteq \mathbb{V}^C$ die Menge aller symbolischen Multimengen eines Universums C.

Die spezielle symbolische Multimenge M_C repräsentiert eine Instanz einer Clafer-Spezifikation $cs \in CS$. Entsprechend gibt die Zählfunktion $m_C(c)$ die Anzahl der Instanzen von Clafer-Definitionen der symbolischen Multimenge M_C an. Alle Symbole $v \in \mathbb{V}$ sind Variablen der Sprache $\mathcal{L}_n$, die durch arithmetische Operationen miteinander verknüpft werden können. In der Sprache $\mathcal{L}_\mathbb{B}$ werden arithmetische Ausdrücke, die in $\mathcal{L}_n$ formuliert sind, durch Vergleichsoperatoren zu atomaren Aussagen zueinander in Beziehung gesetzt und durch aussagenlogische Operationen verknüpft. Zunächst definieren wir die Sprache $\mathcal{L}_n$ zur Beschreibung numerischer Operationen zwischen symbolischen Zählfunktionen.

Definition 4.23 (Sprache zur Beschreibung numerischer Operationen zwischen symbolischer Zählfunktionswerten symbolischer Multimengen) Wir definieren die Sprache $\mathcal{L}_n$ induktiv wie folgt:

- Die Symbole $v \in \mathbb{V}$ sind Variablen, wobei für jeden Funktionswert $c \in C$ einer symbolischen Zählfunktion $m_P(c)$ ein eigenes Symbol existiert.
- Variablen sind arithmetische Ausdrücke.
- Natürliche Zahlen sind Konstanten $0, 1, 2, \ldots$ der Sprache $\mathcal{L}_n$ und arithmetische Ausdrücke.
- Ist A eine Konstante und B eine Variable, dann ist $A \cdot B$ ein arithmetischer Ausdruck.
- Der arithmetische Ausdruck $|M_P| \in \mathcal{L}_n$ repräsentiert die Kardinalität $\sum_{c \in \text{Supp}(M_P)} m_P(c)$ der symbolischen Multimenge M_P.
- Sind A und B arithmetische Ausdrücke, dann sind $A + B$, $\max(A, B)$ und $\min(A, B)$ arithmetische Ausdrücke.

Darauf aufbauend definieren wir nun die Sprache $\mathcal{L}_\mathbb{B}$ zur Beschreibung von Abhängigkeiten zwischen symbolischen Zählfunktionen.

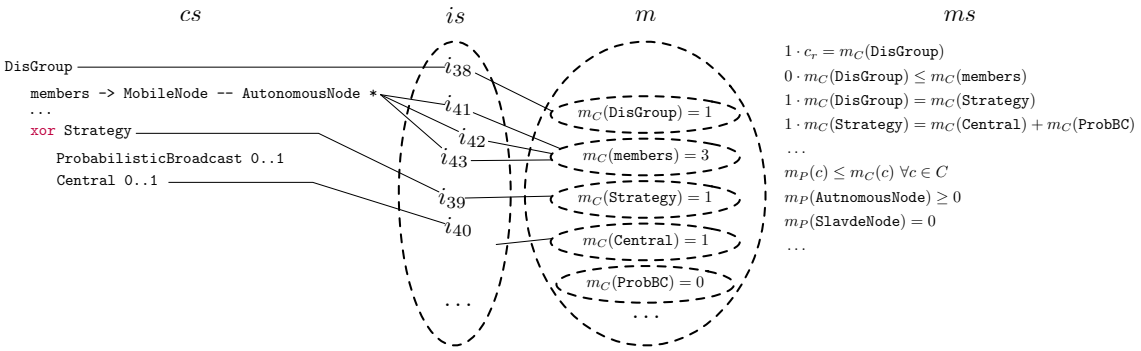


Abbildung 4.8: Zusammenhang zwischen Clafer-Spezifikation, Clafer-Spezifikationsinstanz und Multimengen-Repräsentation

Definition 4.24 (Sprache zur Beschreibung von Abhängigkeiten zwischen Zählfunktionen) Wir definieren die Sprache $\mathcal{L}_{\mathbb{B}}$ induktiv wie folgt:

- Sind A und B arithmetische Ausdrücke (d. h. $A, B \in \mathcal{L}_n$), dann sind $A \leq B$, $A \geq B$, $A < B$, $A > B$ und $A = B$ aussagenlogische Formeln.
- Sind A und B aussagenlogische Formeln, dann sind $\neg A$, $A \wedge B$ und $A \vee B$, $A \implies B$ aussagenlogische Formeln.

Beispiel 4.25 (Zusammenhänge zwischen Clafer-Spezifikationen, Clafer-Spezifikationsinstanzen und Multimengen-Repräsentation) Abbildung 4.8 zeigt den Zusammenhang zwischen einer Clafer-Spezifikation *cs*, der zugehörigen Clafer-Spezifikationsinstanz *is* und einer Multimengen-Belegung *m* sowie der Multimengen-Repräsentation *ms*. Die Multimengen-Repräsentation *ms* besteht aus den symbolischen Multimengen $\mathbb{M}_C$ sowie aus Bedingungen zwischen Zählfunktionswerten der symbolischen Multimengen, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ beschrieben sind. Die Multimengen-Belegung *m* liefert eine Zuweisung der Zählfunktion der Multimenge M_C, die alle Bedingungen der Multimengen-Repräsentation *ms* erfüllt. Im gezeigten Beispiel erfüllt die Zuweisung $m_C(\texttt{members}) = 3$ zusammen mit den anderen Zuweisungen die Bedingungen der Multimengen-Repräsentation. Die Belegung *m* repräsentiert eine gültige Clafer-Spezifikationsinstanz *is*.

Die symbolische Multimenge M_C, weitere Multimengen aus $\mathbb{M}_C$ sowie Bedingungen $L \subseteq \mathcal{L}_{\mathbb{B}}$ zwischen Zählfunktionen, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ formuliert sind, definieren die Multimengen-Repräsentation *ms* einer Clafer-Spezifikation. Die Sprache *MS* repräsentiert die Menge aller Multimengen-Repräsentationen $ms \in MS$.

Definition 4.26 (Multimengen-Repräsentation einer Clafer-Spezifikation) Die Multimengen-Repräsentation einer Clafer-Spezifikation ist definiert als Tupel $ms = (\mathbb{M}_C, L)$. Die Komponenten des Tupels ms sind wie folgt definiert:

- $M_C \in \mathbb{M}_C$ repräsentiert die Instanz einer Clafer-Spezifikation, $\mathbb{M}_C$ ist eine Menge symbolischer Multimengen über C.
- $L \subseteq \mathcal{L}_{\mathbb{B}}$ sind Bedingungen zwischen Zählfunktionswerten der symbolischen Multimengen $\mathbb{M}_C$.

Eine konkrete Multimenge $\mathfrak{M}_P : P \to \mathbb{N}_0$ ist die Abbildung einer Trägermenge $P \subseteq C$ auf die Menge der natürlichen Zahlen $\mathbb{N}_0$. Durch diese Abbildung wird jeder Clafer-Definition eine natürliche Zahl zugeordnet und ermöglicht, eine konkrete Belegung von Variablensymbolen einer symbolischen Multimenge M_P zu repräsentieren. Die Menge aller konkreten Multimengen des Universums C kennzeichnen wir mit $\mathbb{N}_0^C$. Wir können nun Multimengen-Belegungen als Semantik einer Multimengen-Repräsentation definieren.

Definition 4.27 (Semantik einer Multimengen-Repräsentation) Die Semantik einer Multimengen-Repräsentation $ms \in MS$ ist die Menge $[\![ms]\!] := \{\, \mathfrak{M}_C \in \mathbb{N}^C \mid \mathfrak{M}_C \models ms \,\}$ aller konkreten Multimengen, welche die Bedingungen einer Multimengen-Repräsentation ms erfüllen.

Für die Transformation $T^{ms} : CS \to MS$ einer Clafer-Spezifikation $cs \in CS$ in eine Multimengen-Repräsentation $ms \in MS$ fordern wir, dass diese für unsere Analyse *semantikerhaltend* ist. Der Übergang, für den diese Eigenschaft gefordert wird, ist in Abbildung 4.7 mit ❶ markiert. Damit T^{ms} semantikerhaltend ist, müssen folgende beiden Bedingungen gelten:

(i) Für jede Clafer-Spezifikationsinstanz $is \in [\![cs]\!]$ existiert mindestens eine Multimengen-Repräsentation $m \in [\![T^{ms}(cs)]\!]$, für welche die Zählfunktion der Multimengen-Repräsentation genauso viele Instanzen einer Clafer-Definition $c \in C$ angibt wie in der Clafer-Spezifikationsinstanz enthalten sind. Es gilt somit die Bedingung

$$\forall cs \in CS : \forall is \in [\![cs]\!] : \exists m \in [\![T^{ms}(cs)]\!] : \forall c \in C : m_C(c) = |is(c)| \,. \tag{4.1}$$

(ii) Darüber hinaus muss für jede Multimengen-Belegung $m \in [\![T^{ms}(cs)]\!]$ mindestens eine Clafer-Spezifikationsinstanz existieren, für welche genauso viele Instanzen einer Clafer-Definition bestehen wie durch die Zählfunktion der Multimengen-Repräsentation angegeben sind. Dies kann nur für einen Kern CS^k der Sprache Clafer gewährleistet werden, den wir in Abschnitt 6.1 als *Kernsprache* charakterisieren. Es gilt somit zusätzlich die Bedingung

$$\forall cs \in CS^k : \forall m \in [\![T^{ms}(cs)]\!] : \exists is \in [\![cs]\!] : \forall c \in C : |is(c)| = m_C(c). \tag{4.2}$$

Zur Transformation eines Mengenausdrucks `sExp` $\in \langle setExpr \rangle$ in die Multimengen-Repräsentation führen wir folgende Transformationsfunktion ein:

$$T_M^{ms} : \langle setExpr \rangle \to \mathbb{M}_C \times \mathcal{L}_{\mathbb{B}}.$$

Die Transformationsfunktion ordnet einem Mengenausdruck `sExp` eine symbolische Multimenge $M_P \in \mathbb{M}_C$ sowie (symbolische) aussagenlogische Formeln $b \in \mathcal{L}_{\mathbb{B}}$ mit Bedingungen über Zählfunktionen zu.

Um boolesche Ausdrücke `bExp` $\in \langle boolExpr \rangle$ in Clafer-Spezifikationen zu Bedingungen in der Multimengen-Repräsentation zu transformieren, führen wir folgende Transformationsfunktion ein:

$$T_{\mathbb{B}}^{ms} : \langle boolExpr \rangle \to \mathcal{L}_{\mathbb{B}}.$$

Ein boolescher Ausdruck einer Clafer-Spezifikation wird hierbei einer (symbolischen) aussagenlogischen Formel $b \in \mathcal{L}_{\mathbb{B}}$ der Multimengen-Repräsentation zugeordnet, die als atomare Aussagen Bedingungen über Zählfunktionen der Multimengen-Repräsentation enthält. Entsprechend führen wir zur Transformation numerischer Ausdrücke `nExp` $\in \langle numExpr \rangle$ folgende Transformationsfunktion ein:

$$T_n^{ms} : \langle numExpr \rangle \to \mathcal{L}_n.$$

Ein numerischer Ausdruck einer Clafer-Spezifikation wird hierbei einer numerischen Formel $l \in \mathcal{L}_n$ der Multimengen-Repräsentation zugeordnet.

4.3.3 *Repräsentation als mathematisches Optimierungsproblem*

Die Suche nach gültigen Multimengen-Belegungen einer Multimengen-Repräsentation formulieren wir als mathematisches Optimierungsproblem [154]. Zur Formulierung von mathematischen Optimierungsproblemen führen wir die Sprache IP ein. Ein mathematisches Optimierungsproblem $ip \in IP$ besteht aus einer Menge von Ungleichungen über einer Menge von k ganzzahligen oder reellwertigen Entscheidungsvariablen. Die Menge aller Entscheidungsvariablen repräsentieren wir als Vektor $\mathbf{x}$. Wir schreiben $\mathbf{x}(c)$, um auf die Komponente des Vektors zuzugreifen, welche die Zählfunktion $m_C(c)$ codiert. Die Menge aller Ungleichungen lässt sich in der Form $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ repräsentieren, wobei $\mathbf{A}$ eine Matrix und $\mathbf{b}$ ein Vektor mit reellwertigen und ganzzahligen Parametern ist. Die resultierende konvexe Hülle des relaxierten nicht-ganzzahligen Optimierungsproblems beinhaltet den zulässigen Bereich in einem k-dimensionalen Suchraum. Eine Zielfunktion gibt an, ob entweder ein unterer (minimaler) oder oberer (maximaler) Grenzwert für die Entscheidungsvariablen der Zielfunktion durch einen Solver gesucht werden soll. Die Zielfunktion lässt sich als Produkt $\mathbf{z}^T\mathbf{x}$ repräsentieren, wobei $\mathbf{z}^T$ ein transponierter Vektor von reellwertigen und ganzzahligen Parametern ist.

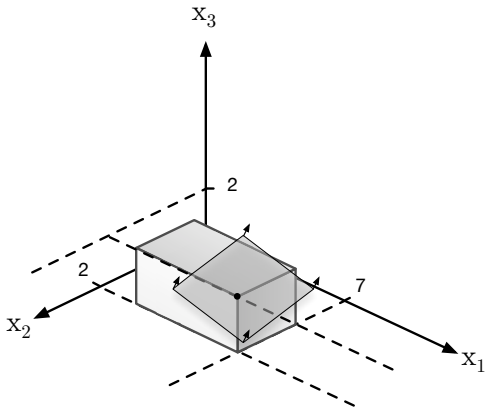
Entscheidungsvariable $\mathbf{x} \in \mathbb{N}_0$

Zielfunktion $\max x_1 + x_2 + x_3$

Nebenbedingungen

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 7 \\ -1 \\ 2 \\ -1 \\ 0 \\ 1 \end{bmatrix}$$

(a) Mathematische Formulierung



(b) Geometrische Interpretation

Abbildung 4.9: Beispiel eines mathematischen Optimierungsproblems

Beispiel 4.28 (Mathematisches Optimierungsproblem) Abbildung 4.9 zeigt das Beispiel eines mathematischen Optimierungsproblems. Die mathematische Formulierung in Abbildung 4.9a enthält die ganzzahligen Entscheidungsvariablen x_1, x_2 und x_3, die Zielfunktion $x_1 + x_2 + x_3$, die maximiert werden soll, sowie Nebenbedingungen in der Form $\mathbf{Ax} \leq \mathbf{b}$. Das mathematische Optimierungsproblem lässt sich geometrisch als Raum interpretieren, der von den Entscheidungsvariablen aufgespannt wird. Die Constraints des relaxierten nicht-ganzzahligen Optimierungsproblems bilden eine konvexe Hülle. Abbildung 4.9b zeigt die geometrische Interpretation des Beispiels.

Definition 4.29 (Mathematisches Optimierungsproblem) Ein mathematisches Optimierungsproblem ist definiert als Tupel $ip = (\mathbf{x}, \mathbf{A}, \mathbf{b}, \mathbf{z})$. Die Sprache IP ist definiert als die Menge aller mathematischen Optimierungsprobleme $ip \in IP$. Die Attribute des Tupels ip sind wie folgt definiert:

- $\mathbf{x}$ ist der Vektor der Entscheidungsvariablen.
- $\mathbf{Ax} \leq \mathbf{b}$ sind die Constraints in Form von Ungleichungen mit Parameter-Matrix $\mathbf{A}$ und Parameter-Vektor $\mathbf{b}$.
- $\mathbf{z}^T\mathbf{x}$ ist die Zielfunktion mit Parameter-Vektor $\mathbf{z}$.

Die Menge aller Lösungen eines mathematischen Optimierungsproblems $ip \in IP$, welche die Constraints erfüllen, bezeichnen wir als $[\![ip]\!]$. Die Menge aller optimalen Lösungen eines mathematischen Optimierungsproblems $ip \in IP$, welche alle Constraints erfüllen sowie die Zielfunktion minimieren bzw. maximieren, bezeichnen wir als $[\![ip]\!]_z$.

Es gilt $[\![ip]\!]_z \subseteq [\![ip]\!]$. Eine Lösung $\mathbf{x} \in [\![ip]\!]$ ist ein Vektor von Zielfunktionswerten. Wir schreiben $\mathbf{x}(c)$, um auf den Wert der Entscheidungsvariablen zu verweisen, die eine Belegung der symbolischen Zählfunktion $m_C(c)$ für Clafer-Definition c repräsentiert.

Definition 4.30 (Lösung des mathematischen Optimierungsproblems) Wir definieren die Menge der *Lösungen eines mathematischen Optimierungsproblems* $[\![ip]\!]$ und die Menge der optimalen Lösungen eines mathematischen Optimierungsproblems $[\![ip]\!]_z$ wie folgt:

- Die Menge aller Lösungen eines mathematischen Optimierungsproblems $ip \in IP$ ist gegeben durch $[\![ip]\!]$. Die Lösung eines mathematischen Optimierungsproblems $\mathbf{x} \in [\![ip]\!]$ ist definiert als Vektor von Zielfunktionswerten, der eine Belegung der Entscheidungsvariablen repräsentiert, die alle Constraints erfüllt (also $A\mathbf{x} \leq \mathbf{b}$).
- Die optimale Lösung eines mathematischen Optimierungsproblems $\mathbf{x}^* \in [\![ip]\!]_z$ mit $[\![ip]\!]_z \subseteq [\![ip]\!]$ ist definiert als Vektor von Zielfunktionswerten, der eine Belegung der Entscheidungsvariablen repräsentiert, die alle Constraints erfüllt und die Zielfunktion $\mathbf{z}^T\mathbf{x}$ minimiert bzw. maximiert.

Für die Transformation $T^{ip} : MS \to IP$ einer Multimengen-Repräsentation $ms \in MS$ in ein mathematisches Optimierungsproblem $ip \in IP$ fordern wir, dass diese für unsere Analyse semantikerhaltend ist. Dieser Übergang ist in Abbildung 4.7 mit ❷ markiert. Sei $ms := T^{ms}(cs)$ das Ergebnis der Transformation einer Clafer-Spezifikation $cs \in CS$ in eine Multimengen-Repräsentation $ms \in MS$, dann muss Folgendes gelten:

(i) Für jede Multimengen-Belegung $m \in [\![ms]\!]$ existiert mindestens eine Lösung des Optimierungsproblems $\mathbf{x} \in T^{ip}(ms)$, für die der Zielfunktionswert und die symbolische Zählfunktion der Multimengen-Repräsentation, die zur identischen Clafer-Definition korrespondieren, gleich sind. Es gilt somit die Bedingung

$$\forall m \in [\![ms]\!] : \exists \mathbf{x} \in [\![T^{ip}(ms)]\!] : \forall c \in C : \mathbf{x}^*(c) = m_C(c).$$

(ii) Darüber hinaus muss für jede Lösung des Optimierungsproblems $\mathbf{x} \in T^{ip}(ms)$ mindestens eine Multimengen-Belegung $m \in [\![ms]\!]$ existieren, für welche die symbolische Zählfunktion der Multimengen-Repräsentation genauso viele Instanzen einer Clafer-Definition $c \in C$ angibt wie der Zielfunktionswert der zur Clafer-Definition c korrespondierenden Entscheidungsvariablen $\mathbf{x}(c)$. Es gilt somit:

$$\forall \mathbf{x} \in [\![T^{ip}(ms)]\!] : \exists m \in [\![ms]\!] : \forall c \in C : m_C(c) = \mathbf{x}^*(c).$$

Die Transformation T^{ip} von Multimengen-Repräsentationen in mathematische Optimierungsprobleme wird in Abschnitt 6.2 im Detail beschrieben.

4.3.4 *Anforderungen an Analyseverfahren*

Für eine Clafer-Spezifikation können wir durch zusätzliche Constraints erzwingen, dass alle Clafer-Spezifikationsinstanzen über eine vorgegebene Anzahl von Instanzen einer Clafer-Definition verfügen. Wir schreiben $cs \oplus \alpha$ für eine entsprechend modifizierte Clafer-Spezifikation cs, die eine Menge zusätzlicher Constraints α erfüllt. Wir fordern, dass die zusätzlichen Constraints selbst Teil der Kernsprache CS^k sind. Auf diese Weise können Clafer-Spezifikationen gezielt auf Inkonsistenzen und Anomalien untersucht werden, wie sie in Abschnitt 3.2.2 beschrieben wurden.

- Falls für eine geforderte Anzahl von n Instanzen einer Clafer-Definition c *keine* Clafer-Spezifikationsinstanz für $cs \oplus \alpha$ gefunden wird, obwohl dies aufgrund der Spezifikation der Multiplizitätsintervalle zulässig ist, liegt für das Multiplizitätsintervall $(n, n) := \lambda_c(c)$ der Clafer-Definition c in cs eine Anomalie vom Typ *totes Kardinalitätsintervall* vor. Es gilt $[\![cs \oplus \alpha]\!] = \emptyset$.
- Falls für eine geforderte beliebig große Anzahl von Instanzen einer Clafer-Definition c eine Clafer-Spezifikationsinstanz für $cs \oplus \alpha$ gefunden wird, ist die Clafer-Definition c in cs unbeschränkt.
- Eine Clafer-Spezifikation cs ist inkonsistent, falls keine Clafer-Spezifikationsinstanz gefunden werden kann und somit $[\![cs]\!] = \emptyset$ gilt.

Beispiel 4.31 (Modifizierte Spezifikation zur Analyse von Multiplizitätsintervallen) Für die in Listing 3.1 gezeigte Spezifikation können wir durch folgenden zusätzlichen Constraint prüfen, ob für die Clafer-Definition `memberOf`, die in der Clafer-Definition `AutonomousNode` geschachtelt ist, eine Anomalie vom Typ *tote Clafer-Definition* vorliegt:

$$\alpha := \{\texttt{[\#(AutonomousNode.memberOf) > 0]}\}$$

Der Constraint wird in die Wurzel-Clafer-Definition c_r geschachtelt. Durch Hinzunahme des gezeigten Constraint prüfen wir für die modifizierte Clafer-Spezifikation $cs \oplus \alpha$, ob mindestens eine Spezifikationsinstanz existiert, in der eine Instanz der Clafer-Definition `memberOf` existiert. Falls für die modifizierte Clafer-Spezifikation $cs \oplus \alpha$ keine Spezifikationsinstanz gefunden werden kann, liegt für die Clafer-Definition `memberOf` eine Anomalie vom Typ *tote Clafer-Definition* vor.

Für die Analyse von Clafer-Spezifikationen auf Inkonsistenzen und Anomalien fordern wir, dass diese sowohl *vollständig* als auch *korrekt* ist. Eine beliebige Clafer-Spezifikation $cs \in CS$ wird für die Analyse zu $cs \oplus \alpha$ modifiziert und in ein mathematisches Optimierungsproblem $ip^\alpha := T^{ip}(T^{ms}(cs \oplus \alpha))$ transformiert.

Das Analyseverfahren ist *vollständig*, falls jede tatsächliche Anomalie oder Inkonsistenz einer Clafer-Spezifikation identifiziert werden kann. Wenn für eine modifizierte Clafer-

Spezifikation $cs \oplus \alpha \in CS$ keine Instanz gefunden wird und demnach die Bedingung $[\![cs \oplus \alpha]\!] = \emptyset$ gilt, darf auch keine Lösung für das zugehörige Optimierungsproblem ip^{α} existieren. Es muss somit die Bedingung $[\![ip^{\alpha}]\!] = \emptyset$ gelten. Aus Gleichung (4.2) folgt, dass unser Analyseverfahren genau dann für die Kernsprache vollständig ist, wenn die Transformation $T^{ip}(T^{ms}(cs \oplus \alpha))$ semantikerhaltend ist. Zusammenfassend ergibt sich folgende Gleichung:

$$\forall cs \oplus \alpha \in CS^{k} : [\![cs \oplus \alpha]\!] = \emptyset \overset{4.2}{\Longrightarrow} [\![ip^{\alpha}]\!] = \emptyset. \tag{4.3}$$

Wir können die Vollständigkeitseigenschaft der Analyse für die Kernsprache CS^{k} von Clafer gewährleisten. Die Kernsprache wird in Abschnitt 6.1 im Detail beschrieben.

Das Analyseverfahren ist *korrekt*, wenn für ein mathematisches Optimierungsproblem ip^{α} einer beliebigen transformierten Clafer-Spezifikation keine Lösung gefunden werden kann, immer auch tatsächlich keine Instanz der zugehörigen Clafer-Spezifikation $cs \oplus \alpha$ existiert. Aus Gleichung (4.1) folgt, dass unser Analyseverfahren für beliebige Clafer-Spezifikationen genau dann korrekt ist, falls die Transformation $T^{ip}(T^{ms}(cs \oplus \alpha))$ semantikerhaltend ist, also folgende Gleichung stets gilt:

$$\forall cs \oplus \alpha \in CS : [\![ip^{\alpha}]\!] = \emptyset \overset{4.1}{\Longrightarrow} [\![cs \oplus \alpha]\!] = \emptyset. \tag{4.4}$$

5

MULTIMENGEN-REPRÄSENTATION VON CLAFER-SPEZIFIKATIONEN

In diesem Kapitel wird die Transformation T^{ms} von Clafer-Spezifikationen in die Multimengen-Repräsentation beschrieben (siehe Abbildung 4.7). Abbildung 5.1 zeigt die Übersicht über die einzelnen Schritte unseres Analyseverfahrens. Zunächst beschreiben wir in Abschnitt 5.1, wie die Vererbungshierarchie von Clafer-Spezifikationen abgeflacht wird, sodass alle abstrakten Clafer-Definitionen aus Clafer-Spezifikationen semantikerhaltend entfernt werden können (siehe ❶ in Abbildung 5.1). In Abschnitt 5.2 beschreiben wir die Transformation der Schachtelungshierarchie einer Clafer-Spezifikation in die Multimengen-Repräsentation (siehe ❷ in Abbildung 5.1). In Abschnitt 5.3 beschreiben wir (i) die Transformation von Referenzen zu Mengenausdrücken und (ii) die Transformation von Referenzen zu numerischen Werten (siehe ❸ Abbildung 5.1). In Abschnitt 5.4 beschreiben wir die Transformation von Pfadausdrücken, die sowohl in Mengenausdrücken als auch innerhalb numerischer Ausdrücke auftreten können (siehe ❹ Abbildung 5.1). In Abschnitt 5.5 beschreiben wir schließlich die Transformation von Ausdrücken der Ausdruckssprache (wie sie Constraints oder Referenzen vorkommen) in die Multimengen-Repräsentation (siehe ❺ in Abbildung 5.1).

5.1 TRANSFORMATION VON VERERBUNGSRELATIONEN

In Clafer stellen Schachtelungs- ($\blacklozenge\!\!\rightarrow$) und Vererbungsrelationen ($\multimap$) orthogonal definierte partielle Ordnungen über der Menge aller Clafer-Definitionen C dar. Falls eine konkrete oder abstrakte Clafer-Definition $c \in C$ von einer abstrakten Clafer-Definition $c' \in C_A$ erbt, muss die Relation $c' \multimap c$ existieren. Die Clafer-Definition c erbt in diesem Fall alle geschachtelten Sub-Clafer-Definitionen und Constraints der abstrakten Clafer-Definition c' sowie aller in der Vererbungshierarchie übergeordneten (abstrakten) Clafer-Definitionen. Instanzen der Clafer-Definition c können somit Instanzen von geerbten Sub-Clafer-Definitionen enthalten. Außerdem müssen für Instanzen von Clafer-Definition c alle geerbten Constraints erfüllt sein.

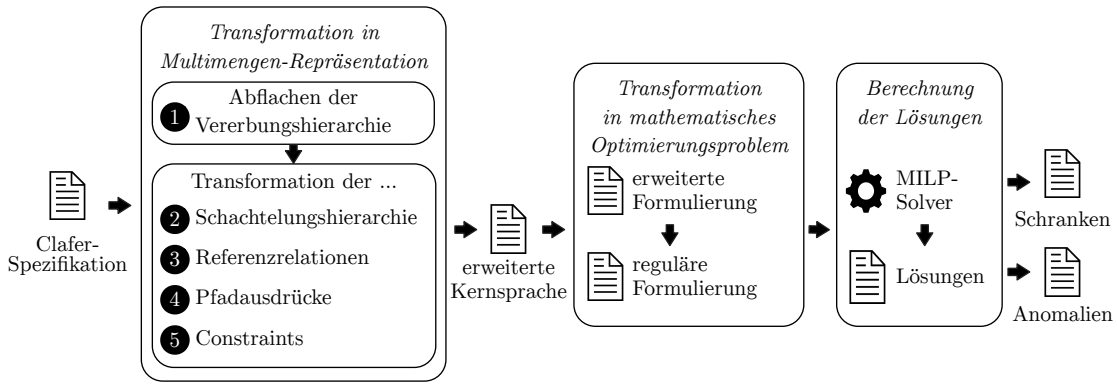


Abbildung 5.1: Übersicht über die einzelnen Schritte unseres Analyseverfahrens

Die Transformation der Vererbungshierarchie einer Clafer-Spezifikation erfolgt in fünf Schritten:

(i) Zunächst werden die eingehenden und ausgehenden Referenzkanten von abstrakten Clafer-Definitionen im Abhängigkeitsgraphen modifiziert. Dies dient als Vorbereitung für die Abflachung der Vererbungshierarchie und der Transformation von Pfadausdrücken, die in Constraints oder Referenzen vorkommen und Bezeichner von abstrakten Clafer-Definitionen enthalten (siehe Abschnitt 5.1.1).

(ii) Im zweiten Schritt wird die Vererbungshierarchie abgeflacht, indem Clafer-Definitionen und Constraints, die in abstrakten Clafer-Definitionen geschachtelt sind, in ihre konkreten Sub-Clafer-Definitionen kopiert werden (siehe Abschnitt 5.1.2).

(iii) Als Nächstes werden auf Basis des zuvor modifizierten Abhängigkeitsgraphen Pfadausdrücke transformiert, die in Mengenausdrücken vorkommen (siehe Abschnitt 5.1.3). Dabei werden Bezeichner von abstrakten Clafer-Definitionen in Pfadausdrücken semantikerhaltend ersetzt. Dies betrifft Pfadausdrücke, die in Mengenausdrücken innerhalb von Referenzen und Constraints vorkommen.

(iv) Im vierten Schritt werden auf Basis des zuvor präparierten Abhängigkeitsgraphen Pfadausdrücke transformiert, die in numerischen Ausdrücken vorkommen (siehe Abschnitt 5.1.4). Dabei werden Bezeichner von abstrakten Clafer-Definitionen semantikerhaltend ersetzt, die in Pfadausdrücken innerhalb numerischer Ausdrücke von Constraints enthalten sind.

(v) Im letzten Schritt werden schließlich die nun redundanten abstrakten Clafer-Definitionen aus der Clafer-Spezifikation entfernt.

In den nachfolgenden Abschnitten beschreiben wir die einzelnen Schritte im Detail.

5.1.1 *Transformation von Referenzrelationen im Abhängigkeitsgraphen*

In diesem Abschnitt beschreiben wir den ersten Schritt der Transformation der Vererbungshierarchie. Dabei werden im Abhängigkeitsgraphen alle eingehenden und ausgehenden Referenzkanten von abstrakten Clafer-Definitionen zu ihren konkreten Sub-Clafer-Definitionen kopiert und anschließend entfernt. Dies dient der Vorbereitung des Abhängigkeitsgraphen für die Abflachung der Vererbungshierarchie (siehe Abschnitt 5.1.2) und für die Transformation von Pfadausdrücken (siehe Abschnitte 5.1.3 und 5.1.4).

Zunächst werden für alle abstrakten Clafer-Definitionen $c_s \in C_A$ die eingehenden Referenzkanten $c \xrightarrow{r} c_s$ zu ihren konkreten Sub-Clafer-Definitionen $c' \in C_O$ mit $c_s \leftarrow^+ c'$ kopiert und anschließend entfernt. Nach Durchführung dieser Modifikation verfügen somit im Abhängigkeitsgraphen alle konkreten Sub-Clafer-Definitionen über die eingehenden Referenzkanten $c \xrightarrow{r} c'$.

Als Nächstes werden für alle abstrakten Clafer-Definitionen $c_s \in C_A$ die ausgehenden Referenzkanten $c_s \xrightarrow{r} c$ zu ihren konkreten Sub-Clafer-Definitionen kopiert und anschließend entfernt. Falls Referenzrelationen durch in der Vererbungshierarchie untergeordnete Clafer-Definitionen verfeinert werden, wird jeweils nur die Referenzkante der untergeordneten Clafer-Definition kopiert, in welcher die Redefinition vorgenommen wird. Nach Durchführung dieser Modifikation existieren somit im Abhängigkeitsgraphen für alle konkreten Sub-Clafer-Definitionen die ausgehenden Referenzkanten $c' \xrightarrow{r} c$. Zusätzlich werden für alle abstrakten Clafer-Definitionen $c_s \in C_A$ die jeweils referenzierten Mengenausdrücke $c_s \twoheadrightarrow \texttt{sExp}$ kopiert, sodass für jede konkrete Sub-Clafer-Definition c', die Relation $c' \twoheadrightarrow \texttt{sExp}$ existiert.

Folgendes Beispiel demonstriert die Modifikation eingehender und ausgehender Referenzkanten von abstrakten Clafer-Definitionen im Abhängigkeitsgraphen.

Beispiel 5.1 (Modifikation von Referenzkanten im Abhängigkeitsgraphen) Die Transformation der Vererbungshierarchie demonstrieren wir in den nachfolgenden Abschnitten anhand der in Abbildung 5.2a gezeigten Beispiel-Clafer-Spezifikation. Die Abbildung 5.2b zeigt den zugehörigen Abhängigkeitsgraphen.

Zunächst werden alle Referenzkanten im Abhängigkeitsgraphen modifiziert, die von konkreten zu abstrakten Clafer-Definitionen zeigen. Dazu werden die Referenzkanten von abstrakten Clafer-Definitionen zu ihren konkreten Sub-Clafer-Definitionen kopiert und anschließend aus dem Abhängigkeitsgraphen entfernt. In der gezeigten Beispiel-Clafer-Spezifikation verfügt die abstrakte Clafer-Definition `h` über eine eingehende Referenzkante `a` $\xrightarrow{r}$ `h`. Die Clafer-Definitionen `i` und `j` sind konkrete Sub-Clafer-Definitionen, die von der abstrakten Clafer-Definition `h` erben. Entsprechend verfügt die abstrakte Clafer-Definition `a` über eine eingehende Referenzkante `b` $\xrightarrow{r}$ `a`, wobei die Clafer-Definitionen `d` und `e` Sub-Clafer-Definitionen

der Clafer-Definition `a` sind. Es werden entsprechend folgende Änderungen am Abhängigkeitsgraphen vorgenommen:

- `a` $\xrightarrow{r}$ `h` wird durch `a` $\xrightarrow{r}$ `i` und `a` $\xrightarrow{r}$ `j` ersetzt.
- `b` $\xrightarrow{r}$ `a` wird durch `b` $\xrightarrow{r}$ `d` und `b` $\xrightarrow{r}$ `e` ersetzt.

Im nächsten Schritt werden alle Referenzkanten modifiziert, die von abstrakten Clafer-Definitionen ausgehen. Nach dem vorherigen Schritt, bei dem eingehende Referenzkanten abstrakter Clafer-Definitionen modifiziert wurden, verfügt die Clafer-Definition `a` nun über zwei ausgehende Referenzkanten zu Clafer-Definition `i` und `j`. Es werden somit folgende Referenzkanten dem Abhängigkeitsgraphen hinzugefügt bzw. ersetzt:

- `a` $\xrightarrow{r}$ `i` wird durch `d` $\xrightarrow{r}$ `i` und `e` $\xrightarrow{r}$ `i` ersetzt.
- `a` $\xrightarrow{r}$ `j` wird durch `d` $\xrightarrow{r}$ `j` und `e` $\xrightarrow{r}$ `j` ersetzt.

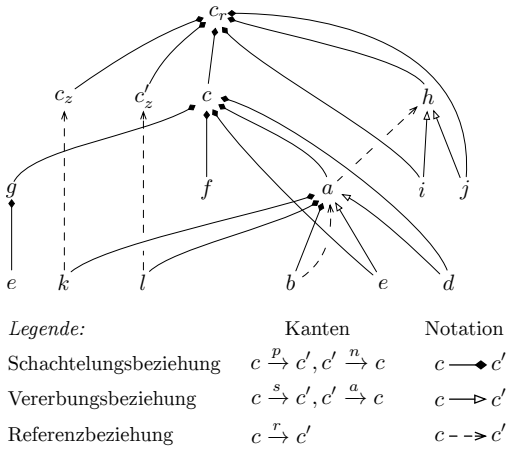
Abbildung 5.3 zeigt den Abhängigkeitsgraphen nach den genannten Modifikationen der Referenzkanten. Zusätzlich werden alle Referenzrelationen $c_s \twoheadrightarrow$ `sExp` zwischen abstrakten Clafer-Definitionen c_s und Mengenausdrücken `sExp` durch die Referenzrelationen $c \twoheadrightarrow$ `sExp` ersetzt, wobei c eine konkrete Sub-Clafer-Definition von der abstrakten Clafer-Definition c_s ist. Im betrachteten Beispiel wird somit die Referenzrelation `a` $\twoheadrightarrow$ `h` durch `d` $\twoheadrightarrow$ `h` und `e` $\twoheadrightarrow$ `h` ersetzt. Der Mengenausdruck, der nun von der konkreten Clafer-Definitionen `d` und `e` referenziert wird, enthält noch den Bezeichner der abstrakten Clafer-Definition `h`. In Abschnitt 5.1.3 zeigen wir, wie auf Basis des modifizierten Abhängigkeitsgraphen die abstrakten Bezeichner in Pfadausdrücken semantikerhaltend durch Bezeichner konkreter Sub-Clafer-Definitionen ersetzt werden.

5.1.2 *Abflachung der Vererbungshierarchie*

Vererbungsrelationen sind in Clafer endliche Wurzelbäume. Eine abstrakte Clafer-Definition kann semantisch äquivalent ersetzt werden, indem die in ihr geschachtelten (konkreten) Clafer-Definitionen und Constraints in die konkreten Clafer-Definitionen kopiert werden, die von der abstrakten Clafer-Definition erben. Als Nächstes beschreiben wir daher, wie auf Basis des zuvor modifizierten Abhängigkeitsgraphen die Vererbungshierarchie abgeflacht wird. Algorithmus 1 beschreibt die Vorgehensweise zur Abflachung der Schachtelungshierarchie (engl. flattening). Der Algorithmus wurde in abgewandelter Form in [147] präsentiert. Durch den Algorithmus wird die Schachtelungshierarchie unterhalb abstrakter Clafer-Definitionen $c \in C_A$ einschließlich geschachtelter Constraints repliziert und in ihre konkreten Sub-Clafer-Definitionen kopiert.

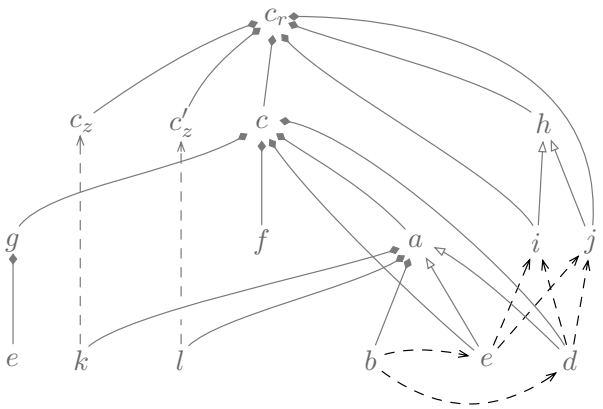
```
c 0..*
  f 0..1
    [some this.parent.g.e]
  abstract g 0..*
    e 0..*
  abstract a -> h 0..*
    l -> integer 0..*
    k -> integer 0..*
    b -> c.a 0..*
  d : a 0..*
  e : a 0..*
  [ this.a.l.dref > this.a.k.dref ]
  [ some this.a.b.dref.b.dref ]
abstract h 0..*
i : h 0..*
j : h 0..*
```

(a) Clafer-Spezifikation



(b) Abhängigkeitsgraph

Abbildung 5.2: Beispiel-Clafer-Spezifikation zur Demonstration der Abflachung von Vererbungshierarchien



Legende:	Kanten	Notation[(*)]
Schachtelungsbeziehung	$c \xrightarrow{p} c', c' \xrightarrow{n} c$	$c \longrightarrow\!\!\blacklozenge\, c'$
Vererbungsbeziehung	$c \xrightarrow{s} c', c' \xrightarrow{a} c$	$c \longrightarrow\!\!\triangleright\, c'$
Referenzbeziehung	$c \xrightarrow{r} c'$	$c \dashrightarrow c'$

*Hervorgehoben falls in letztem Transformationsschritt hinzugefügt

Abbildung 5.3: Abhängigkeitsgraph der Beispiel-Clafer-Spezifikation aus Abbildung 5.2b nach Transformation der Referenzbeziehungen (Schritt 1)

Algorithmus 1 Flattening-Algorithmus zur Abflachung der Vererbungshierarchie

1: flattenSubtree(c_r)

2: **procedure** flattenSubtree(c)
3: | **if** $c \notin C_A$ **then**
4: | | **for all** $c_s \in \{ c_s \in C \mid c_s \leftarrow^+ c \}$ **do**
5: | | | Kopiere Constraint $\Phi(c_s, \texttt{bExp}')$, sodass Relation $\Phi(c, \texttt{bExp}')$ existiert.
6: | | | **for all** $c' \in \{ c' \in C \mid c_s \blacklozenge\!\!\rightarrow c' \}$ **do**
7: | | | | $c_{copy} \leftarrow$ deepCopy(c')
8: | | | | Füge Kanten $c \xrightarrow{n} c_{copy}$ und $c_{copy} \xrightarrow{p} c$ hinzu, sodass Relation $c \blacklozenge\!\!\rightarrow c_{copy}$ existiert.
9: | | **for all** $c' \in \{c' \in C \mid c \blacklozenge\!\!\rightarrow c'\}$ **do**
10: | | | flattenSubtree(c')

11: **procedure** deepCopy(c)
12: | $c_{copy} \leftarrow$ erstelle flache Kopie von Clafer-Definition c
13: | **for all** $c' \in \{ c' \in C \mid c \blacklozenge\!\!\rightarrow c' \}$ **do**
14: | | $c'' \leftarrow$ deepCopy(c')
15: | | Füge Kanten $c_{copy} \xrightarrow{n} c''$ und $c'' \xrightarrow{p} c_{copy}$ hinzu, sodass Relation $c_{copy} \blacklozenge\!\!\rightarrow c''$ existiert.
16: | ▷ Kopiere alle ausgehenden Referenzkanten:
17: | **for all** $c' \in \{ c' \mid c \xrightarrow{r} c' \}$ **do**
18: | | Kopiere $c \twoheadrightarrow \texttt{sExp}$, sodass Relation $c_{copy} \twoheadrightarrow \texttt{sExp}$ existiert.
19: | | Füge Kante $c_{copy} \xrightarrow{r} c'$ hinzu.
20: | ▷ Kopiere alle eingehenden Referenzkanten:
21: | **for all** $c' \in \{ c' \twoheadrightarrow \texttt{sExp} \wedge c \in \psi(c') \}$ **do**
22: | | Füge Kante $c' \xrightarrow{r} c_{copy}$ hinzu.
23: | ▷ Kopiere ausgehende Vererbungsrelation:
24: | **if** $c \notin \{ c' \in C \mid c_s \twoheadleftarrow c \wedge c_s \blacklozenge\!\!\rightarrow^+ c' \}$ **then** ▷ Falls c nicht transitiv in c_s geschachtelt ist.
25: | | Füge Kanten $c_{copy} \xrightarrow{s} c_s$ und $c_s \xrightarrow{a} c_{copy}$ hinzu, sodass Relation $c_s \twoheadleftarrow c_{copy}$ existiert.
26: | **return** c_{copy}

Der Flattening-Algorithmus besteht aus den beiden Prozeduren FLATTENSUBTREE und DEEPCOPY. Die Prozedur FLATTENSUBTREE traversiert ausgehend von der Wurzel-Clafer-Definition c_r (siehe Zeile 1 in Algorithmus 1) die Schachtelungshierarchie aller konkreten Clafer-Definitionen. Die Prozedur FLATTENSUBTREE erwartet als Parameter eine Clafer-Definition c (siehe Zeile 2 in Algorithmus 1), welche die gerade prozessierte Clafer-Definition kennzeichnet. Falls eine Clafer-Definition c nicht abstrakt ist (siehe Zeile 3 in Algorithmus 1), werden von allen in der Vererbungshierarchie übergeordneten Clafer-Definitionen $c_s \twoheadleftarrow^{+} c$ die jeweils geschachtelten Constraints in die Clafer-Definition c kopiert. Zusätzlich werden alle geschachtelten Clafer-Definitionen $c_s \blacklozenge\!\!\rightarrow c'$ durch die Prozedur DEEPCOPY repliziert und in die Clafer-Definition c_{copy} kopiert (siehe Zeile 7 in Algorithmus 1). Das beschriebene Vorgehen wird für alle in Clafer-Definition c geschachtelte Clafer-Definitionen durch rekursiven Aufruf der Prozedur FLATTENSUBTREE wiederholt (siehe Zeile 10 in Algorithmus 1).

Von besonderer Bedeutung ist die Prozedur DEEPCOPY (siehe Zeile 11 in Algorithmus 1), die für eine zu kopierende Clafer-Definition c zunächst eine flache Kopie c_{copy} einschließlich der Multiplizitätsintervalle λ_c und Gruppenkardinalitätsintervalle λ_g anfertigt (siehe Zeile 12 in Algorithmus 1). Anschließend werden rekursiv die in Clafer-Definition c geschachtelten Clafer-Definitionen c' kopiert (siehe Zeile 14 in Algorithmus 1 und über zusätzliche Kanten $c_{copy} \xrightarrow{n} c''$ in Clafer-Definition c_{copy} geschachtelt (siehe Zeile 15 in Algorithmus 1).

Eingehende und ausgehende Referenzkanten der zu kopierenden Clafer-Definition c werden durch die Prozedur DEEPCOPY wie folgt behandelt: Zunächst werden für alle ausgehenden Referenzkanten $c \xrightarrow{r} c'$ im Abhängigkeitsgraphen zusätzliche Referenzkanten $c_{copy} \xrightarrow{r} c'$ hinzugefügt (siehe Zeile 18 in Algorithmus 1). Entsprechend wird für alle eingehende Referenzkanten vorgegangen (siehe Zeile 21 bis 22 in Algorithmus 1). Außerdem wird für jede Referenzrelation $c \twoheadrightarrow \texttt{sExp}$ eine Referenzrelation $c_{copy} \twoheadrightarrow \texttt{sExp}$ hinzugefügt (siehe Zeile 19 in Algorithmus 1).

Um zyklische Abhängigkeiten zwischen Vererbungs- und Schachtelungsrelationen aufzubrechen, werden ausgehende Vererbungskanten $c \xrightarrow{s} c_s$ nur dann kopiert und dem Abhängigkeitsgraphen hinzugefügt, wenn die zu kopierende Clafer-Definition c nicht selbst transitiv in Clafer-Definition c_s geschachtelt ist (siehe Zeile 24 bis 25 in Algorithmus 1).

In folgendem Beispiel demonstrieren wir die Abflachung der Vererbungshierarchie mithilfe der Beispiel-Clafer-Spezifikation in Abbildung 5.2a.

Beispiel 5.2 (Abflachung der Vererbungshierarchie) Abbildung 5.2a zeigt die Beispiel-Clafer-Spezifikation mit einer mehrstufigen Vererbungs- und Schachtelungshierarchie, die im Folgenden abgeflacht werden soll. Wir nehmen an, dass eingehende und ausgehende Referenzkanten abstrakter Clafer-Definitionen im Abhängigkeitsgraphen bereits gemäß Abschnitt 5.1.1 präpariert wurden, sodass der in Abbildung 5.3

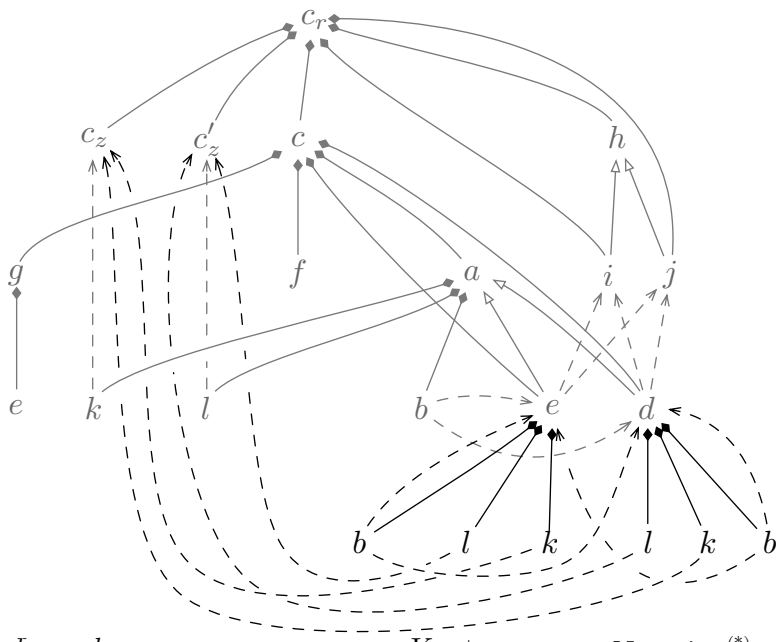
gezeigte Abhängigkeitsgraph als Eingabe des aktuellen Schritts dient. Der Abhängigkeitsgraph nach Durchführung der Abflachung der Schachtelungshierarchie ist in Abbildung 5.4 gezeigt.

Algorithmus 1 beginnt ausgehend von Clafer-Definition c_r, durch den rekursiven Aufruf der Prozedur FlattenSubtree die Schachtelungshierarchie konkreter Clafer-Definitionen zu traversieren (siehe Zeile 9 bis 10 in Algorithmus 1). Falls eine gerade prozessierte konkrete Clafer-Definition eine abstrakte und in der Vererbungshierarchie übergeordnete Clafer-Definition aufweist, werden alle darin geschachtelten Clafer-Definitionen sowie Constraints repliziert und zu der gerade prozessierten Clafer-Definition kopiert (siehe Zeile 5 bis 8 in Algorithmus 1). Im betrachteten Beispiel erben die konkreten Clafer-Definitionen `d` und `e` von der abstrakten Clafer-Definition `a`. Die Clafer-Definitionen `b`, `l` und `k` sind wiederum in der abstrakten Clafer-Definition `a` geschachtelt. Der Flattening-Algorithmus repliziert daher mithilfe der Prozedur DeepCopy die Clafer-Definitionen `b`, `l` und `k`, und fügt sie der Schachtelungshierarchie unterhalb der Clafer-Definitionen `d` und `e` hinzu (siehe Zeile 6 in Algorithmus 1). Nach Durchführung der Abflachung der Vererbungshierarchie enthält der Abhängigkeitsgraph zwei zusätzliche Clafer-Definitionen mit den Bezeichnern `d` sowie zwei weitere Clafer-Definitionen mit den Bezeichnern `e`. Es ergibt sich somit der in Abbildung 5.4 gezeigte Abhängigkeitsgraph, der als Zwischenrepräsentation für die nachfolgende Transformation der Pfadausdrücke (mit abstrakten Bezeichnern) dient.

5.1.3 *Abflachen von Pfadausdrücken in Mengenausdrücken*

Als Nächstes betrachten wir die Transformation von Pfadausdrücken, deren entsprechende Pfade abstrakte Clafer-Definitionen im Abhängigkeitsgraphen traversieren. Derartige Pfadausdrücke können innerhalb von Mengenausdrücken $\texttt{sExp} \in \langle setExpr \rangle$ vorkommen, die entweder Bestandteil von Referenzen $c \twoheadrightarrow \texttt{sExp}$ oder Constraints $\Phi(c_{ctx}, \texttt{bExp})$ sind. Durch den im letzten Abschnitt vorgestellten Flattening-Algorithmus werden Constraints $\Phi(c_s, \texttt{bExp})$ mit abstrakter Kontext-Clafer-Definition $c_s \in C_A$ kopiert und in jeder konkreten Sub-Clafer-Definition $c \in C_O$ mit $c_s \blacklozenge\!\!\rightarrow^{+} c$ geschachtelt.

Die konkreten Sub-Clafer-Definitionen einer abstrakten Clafer-Definition $c_s \in C_A$ entsprechen den Blättern des Teilbaums, der sich durch die Vererbungsrelation $\twoheadleftarrow$ über die Menge C ergibt und in c_s verwurzelt ist. Instanzen einer abstrakten Clafer-Definition sind demnach immer auch Instanzen der konkreten Sub-Clafer-Definition dieses Teilbaums. Da konkrete Clafer-Definitionen nur von abstrakten Clafer-Definitionen erben, bilden Instanzen konkreter Clafer-Definitionen disjunkte Teilmengen. Dies bedeutet, dass eine Instanz einer konkreten Clafer-Definition $c \in C_O$ nicht gleichzeitig Instanz einer anderen konkreten Clafer-Definition $c' \in C_O$ sein kann. Eine abstrakte Clafer-Definition kann somit als Vereinigung der Mengen aller Instanzen der erbenden konkreten Sub-



Legende:	Kanten	Notation(*)
Schachtelungsbeziehung	$c \xrightarrow{p} c', c' \xrightarrow{n} c$	$c \longrightarrow\!\!\blacklozenge\, c'$
Vererbungsbeziehung	$c \xrightarrow{s} c', c' \xrightarrow{a} c$	$c \longrightarrow\!\!\triangleright\, c'$
Referenzbeziehung	$c \xrightarrow{r} c'$	$c \dashrightarrow c'$

*Hervorgehoben falls in letztem Transformationsschritt hinzugefügt

Abbildung 5.4: Abhängigkeitsgraph der Beispiel-Clafer-Spezifikation aus Abbildung 5.2b nach Anwendung des Flattening-Algorithmus (Schritt 2)

Clafer-Definitionen repräsentiert werden. Demzufolge ersetzen wir jedes Vorkommen eines Pfades p, der eine abstrakte Clafer-Definition $c' \in C_A$ traversiert, innerhalb von Mengenausdrücken $\texttt{sExp} \in \langle setExpr \rangle$ durch die Vereinigung $p' \texttt{++} \ \ldots \ \texttt{++} \ p''$ der transformierten Pfade $p', \ldots, p''$. Für die transformierten Pfade $p', \ldots, p''$ fordern wir, dass diese ausschließlich konkrete Clafer-Definitionen im Abhängigkeitsgraphen traversieren. Im Folgenden beschreiben wir, wie ein Pfadausdruck $\texttt{pExpr} \in \langle pathExpr \rangle$, der durch einen Pfad p repräsentiert wird (siehe Definition 4.12), und potentiell abstrakte Clafer-Definitionen im Abhängigkeitsgraphen traversiert, auf Basis des zuvor präparierten Abhängigkeitsgraphen in eine Menge von Pfaden transformiert wird, die ausschließlich konkrete Clafer-Definitionen traversieren.

In Abschnitt 4.2.1 haben wir gezeigt, dass ein Pfad zu mehreren Kantenfolgen im Abhängigkeitsgraphen aufgelöst werden kann. Eine Kantenfolge entspricht hierbei einer Sequenz von Kanten, die beim Auswerten eines Pfades durchlaufen werden. Wir definieren die *Auswertungskantenfolge* eines Pfades, die aus dem Traversieren eines Pfades p im Abhängigkeitsgraphen resultiert, wie folgt:

Definition 5.3 (Auswertungskantenfolge eines Pfades) Sei $\texttt{pExpr} \in \langle pathExpr \rangle$ ein Pfadausdruck eines Pfades $p = (id_1, \ldots, id_n)$ für eine Kontext-Clafer-Definition $c_{ctx} \in C$. Die Auswertungskantenfolge zu einem Pfad p ist definiert als Sequenz $w = (c \xrightarrow{t} c' \xrightarrow{t} \ldots \xrightarrow{t} c'')$. Die Elemente der Sequenz sind Kanten des Abhängigkeitsgraphen. Die Clafer-Definition c'' ist ausgehend von einer Kontext-Clafer-Definition c_{ctx} durch schrittweises Traversieren des Abhängigkeitsgraphen gemäß p erreichbar. Falls ein Pfad kontextabhängig ist, gilt $c = c_{ctx}$.

Algorithmus 2 liefert für einen Pfad p, der potentiell abstrakte Clafer-Definitionen im Abhängigkeitsgraphen traversiert, die Menge aller möglichen Auswertungskantenfolgen, die nur konkrete Clafer-Definitionen des modifizierten Abhängigkeitsgraphen durchlaufen. Wir fordern vor Anwendung des Algorithmus, dass der Abhängigkeitsgraph nach dem in den Abschnitten 5.1.1 und 5.1.2 gezeigten Vorgehen präpariert wurde, sodass eingehende und ausgehende Referenzkanten und geschachtelte Clafer-Definitionen sowie Constraints abstrakter Clafer-Definitionen in ihre konkreten Sub-Clafer-Definitionen kopiert wurden.

Im Folgenden beschreiben wir Algorithmus 2 genauer. Der Algorithmus beginnt mit der Initialisierung der Datenstrukturen (siehe Zeile 1 bis 2 in Algorithmus 2): In einem ersten Schritt werden die Menge der gefundenen Auswertungskantenfolgen W als leere Menge und die aktuell traversierte Auswertungskantenfolge w als leere Sequenz initialisiert. Als Nächstes wird die Prozedur RESOLVE rekursiv für eine konkrete Kontext-Clafer-Definition $c_{ctx} \in C_O$ und einen zu transformierenden Pfad p aufgerufen (siehe Zeile 3 in Algorithmus 2). Die Prozedur RESOLVE wertet schrittweise einen noch zu prozessierenden Pfad p_{cur} aus, indem rekursiv in Abhängigkeit vom ersten Element eines Pfades $id_{next} = p_{cur}(1)$ (siehe Zeile 6 in Algorithmus 2) der Abhängigkeitsgraph traversiert wird.

Algorithmus 2 Ermittlung der Auswertungskantenfolgen von Pfaden

$W = \emptyset$ ▷ Initialisiere Menge aller Auswertungskantenfolgen als leere Menge.
$w = [\,]$ ▷ Initialisiere aktuelle Auswertungskantenfolge als leere Sequenz.
RESOLVE(c_{ctx}, p, p) ▷ Beginne mit Auswertung.

procedure RESOLVE(c_{cur}, p_{cur}, p_{orig})
  $id_{next} \leftarrow p_{cur}(1)$
  if $id_{next} = \bot$ **then**
    Füge aktuellen Auswertungskantenfolge w der Menge W hinzu.
  else if $p_{cur} = p_{orig} \wedge id_{next} \neq$ '`this`' **then** ▷ Löse global auf.
    $c \leftarrow$ UNIQUE($\{\, c \mid c_r \blacklozenge\!\!\rightarrow c \wedge id(c) = id_{next} \,\}$)
    if $c \in C_A$ **then**
      for all $c' \in \{\, c' \in C \setminus C_A \mid c \leftarrow\!\!\blacklozenge^{+} c' \,\}$ **do**
        $w \leftarrow$ Kanten der Schachtelungshierarchie von c_r zu c'
        RESOLVE(c', DROPFIRST(p_{cur}), p_{orig})
        Entferne alle Kanten aus w.
    else
      Füge Kante $c_r \xrightarrow{n} c$ am Ende der Sequenz w hinzu.
      RESOLVE(c, DROPFIRST(p_{cur}), p_{orig})
      Entferne letzte Kante von w.
  else if $id_{next} =$ '`this`' **then** ▷ Löse Kontext-Clafer-Definition auf.
    RESOLVE(c_{cur}, DROPFIRST(p_{cur}), p_{orig})
    Entferne letzte Kante von w.
  else if $id_{next} =$ '`parent`' **then** ▷ Löse inverse Schachtelungsbeziehung auf.
    $c_p \leftarrow$ UNIQUE($\{\, c_p \mid c_p \blacklozenge\!\!\rightarrow c \,\}$)
    Füge Kante $c_{cur} \xrightarrow{p} c_p$ am Ende der Sequenz w hinzu.
    RESOLVE(c_p, DROPFIRST(p_{cur}), p_{orig})
    Entferne letzte Kante von w.
  else if $id_{next} =$ '`dref`' **then** ▷ Löse `dref` auf.
    for all $c \in \{\, c \mid c_{cur} \xrightarrow{r} c \wedge \neg(\exists c_s \in C_A : c_s \blacklozenge\!\!\rightarrow^{+} c) \,\}$ **do**
      Füge Kante $c_{cur} \xrightarrow{r} c$ am Ende der Sequenz w hinzu.
      RESOLVE(c, DROPFIRST(p_{cur}), p_{orig})
      Entferne letzte Kante von w.
  else ▷ Löse Schachtelungsbeziehung auf.
    $c \leftarrow$ UNIQUE($\{\, c \mid c_{cur} \blacklozenge\!\!\rightarrow c \wedge id(c) = id_{next} \,\}$)
    if $c \in C_A$ **then**
      for all $c' \in \{\, c' \in C \setminus C_A \mid c \leftarrow\!\!\blacklozenge^{+} c' \,\}$ **do**
        ▷ Abbruch und $W = \emptyset$, falls kein c' existiert.
        Füge Kante $c_{cur} \xrightarrow{n} c'$ dem Ende der Sequenz w hinzu.
        RESOLVE(c', DROPFIRST(p_{cur}), p_{orig})
        Entferne letzte Kante von w.
    else
      Füge Kante $c_{cur} \xrightarrow{n} c$ am Ende der Sequenz w hinzu.
      RESOLVE(c, DROPFIRST(p_{cur}), p_{orig})
      Entferne letzte Kante von w.

Hierbei werden der Auswertungskantenfolge w die durch den Algorithmus traversierten Kanten hinzugefügt. Falls der noch zu prozessierende Pfad p_{cur} kein weiteres Element enthält, ist dieser vollständig aufgelöst bis zu einem Element c_{cur}. In diesem Fall wird der Menge aller gefundenen Auswertungskantenfolgen W die Auswertungskantenfolge w hinzugefügt (siehe Zeile 7 in Algorithmus 2).

Falls der Pfad kontextunabhängig ist – das erste Element des Pfades ist in diesem Fall also nicht **this** – (siehe Zeile 8 in Algorithmus 2), wird auf der obersten Ebene der Schachtelungshierarchie (unterhalb der Clafer-Definition c_r) nach einer eindeutigen Clafer-Definition c gesucht, mit $id_{next} = id(c)$ (siehe Zeile 9 in Algorithmus 2). Wenn die gefundene Clafer-Definition c nicht abstrakt ist, wird der Pfad ausgehend von Clafer-Definition c weiter traversiert (siehe Zeile 16 bis 18 in Algorithmus 2). Falls die gefundene Clafer-Definition c hingegen abstrakt ist (siehe Zeile 10 in Algorithmus 2), dann werden für alle konkreten Sub-Clafer-Definitionen, die transitiv von Clafer-Definition c erben, die Kanten von der Wurzel-Clafer-Definition c_r bis zu ihrer konkreten Position in der Schachtelungshierarchie dem Kantenzug w hinzugefügt (siehe Zeile 12 in Algorithmus 2). Dies ist erforderlich, da konkrete Sub-Clafer-Definitionen von c potentiell über die Schachtelungshierarchie verteilt sind. Anschließend wird der Abhängigkeitsgraph ausgehend von jeder gefundenen konkreten Sub-Clafer-Definition weiter traversiert (siehe Zeile 12 in Algorithmus 2).

Falls der Pfad kontextabhängig ist, das erste Element des Pfades also das Schlüsselwort **this** ist (siehe Zeilen 19 in Algorithmus 2), wird der Pfad ausgehend von Clafer-Definition c_{ctx} traversiert. Hierzu wird durch die Hilfsprozedur DROPFIRST das erste Element des Pfades p_{cur} entfernt und die Methode RESOLVE darauf rekursiv aufgerufen (siehe Zeile 20 in Algorithmus 2).

Falls das nächste Element eines Auswertungsschrittes das Schlüsselwort **parent** ist (siehe Zeile 22 in Algorithmus 2), wird ausgehend von Clafer-Definition c_{cur} die Kante $c_{cur} \xrightarrow{p} c_p$ der Auswertungskantenfolge w hinzugefügt (siehe Zeile 24 in Algorithmus 2) und ausgehend von Clafer-Definition c_p weiter durch den Abhängigkeitsgraphen traversiert (siehe Zeile 25 in Algorithmus 2).

Falls das nächste Element eines Auswertungsschrittes das Schlüsselwort **dref** ist (siehe Zeile 27 in Algorithmus 2), wird der Abhängigkeitsgraph weiter über alle ausgehenden Referenzkanten $c_{cur} \xrightarrow{r} c$ traversiert, die nicht auf Clafer-Definitionen zeigen, die in abstrakten Clafer-Definitionen geschachtelt sind (siehe Zeile 28 in Algorithmus 2). Der Abhängigkeitsgraph wird somit nur über solche Referenzkanten traversiert, die auf zuvor kopierte konkrete Sub-Clafer-Definitionen zeigen. Durch die in Abschnitt 5.1.1 gezeigten Modifikationen ist sichergestellt, dass keine Referenzkanten zu abstrakten Clafer-Definitionen existieren.

Falls das nächste Element eines Auswertungsschrittes der Bezeichner einer Clafer-Definition ist, wird nach einer Clafer-Definition c gesucht, die in der Clafer-Definition c_{cur} geschachtelt ist, und für welche die Bedingung $id(c) = id_{next}$ gilt (siehe Zeile 33 in Algorithmus 2). Falls die gefundene Clafer-Definition c abstrakt ist (siehe Zeile 34 in

Algorithmus 2), wird der Abhängigkeitsgraph ausgehend von c_{cur} über die Kanten $c_{cur} \xrightarrow{n} c'$ weiter traversiert, wobei c' alle konkreten Sub-Clafer-Definitionen sind, die von Clafer-Definition c erben. Durch die Wohlgeformtheitseigenschaften zwischen Schachtelungs- und Vererbungsrelationen, die in Abschnitt 4.2.2 vorgestellt wurden, ist sichergestellt, dass eine konkrete Clafer-Definition auf der gleichen Ebene der Schachtelungshierarchie existieren muss, soweit mindestens eine Clafer-Spezifikationsinstanz existiert, in welcher der Pfadausdruck keine leere Instanzmenge referenziert. Falls keine konkrete Sub-Clafer-Definition c' in der Clafer-Spezifikation existiert, wird die Auswertung abgebrochen und der Menge aller Auswertungskantenfolge W die leere Menge zugewiesen. Ist c nicht abstrakt, wird der Abhängigkeitsgraph ausgehend von der Clafer-Definition c über die Kante $c_{cur} \xrightarrow{n} c'$ traversiert (siehe Zeile 41 in Algorithmus 2).

Algorithmus 2 liefert somit für einen Pfad p eine Menge von Auswertungskantenfolgen W. Diese werden im nächsten Schritt zu Pfadausdrücken umgewandelt und durch eine Mengenvereinigung miteinander verknüpft. Die Vereinigung der transformierten Pfadausdrücke ersetzt den ursprünglichen Pfadausdruck. Falls die Menge der Auswertungskantenfolgen leer ist, wird eine zusätzliche Hilfs-Clafer-Definition `emptyset` unterhalb der Wurzel-Clafer-Definition c_r eingeführt, mit dem Multiplizitätsintervall $\lambda_c(\texttt{emptyset}) = (0, 0)$, das den ursprünglichen Pfad p ersetzt.

In folgendem Beispiel demonstrieren wir die Abflachung von Pfaden, die in Mengenausdrücken auftreten und abstrakte Clafer-Definitionen traversieren.

Beispiel 5.4 (Abflachen von Pfaden innerhalb von Mengenausdrücken) Wir betrachten folgenden Ausschnitt der Beispiel-Clafer-Spezifikation, die in Abbildung 5.2a abgebildet ist:

```
c
  [ some this.a.b.dref.b.dref]
```

Der im gezeigten Ausschnitt enthaltene Constraint ist in Kontext-Clafer-Definition `c` geschachtelt. Die Auswertungskantenfolge des Pfadausdrucks ergibt sich im ursprünglichen unmodifizierten Abhängigkeitsgraphen wie folgt:

$$c \xrightarrow{n} a \xrightarrow{n} b \xrightarrow{r} a \xrightarrow{n} b \xrightarrow{r} a.$$

Durch den Pfad wird demnach die abstrakte Clafer-Definition `a` traversiert, sodass der Pfadausdruck abgeflacht werden muss. Algorithmus 2 löst den Pfadausdruck zu folgenden Auswertungskantenfolgen auf:

- $c \xrightarrow{n} d \xrightarrow{n} b \xrightarrow{r} d \xrightarrow{n} b \xrightarrow{r} d$,
- $c \xrightarrow{n} d \xrightarrow{n} b \xrightarrow{r} d \xrightarrow{n} b \xrightarrow{r} e$,
- $c \xrightarrow{n} d \xrightarrow{n} b \xrightarrow{r} e \xrightarrow{n} b \xrightarrow{r} d$,

- $c \xrightarrow{n} d \xrightarrow{n} b \xrightarrow{r} e \xrightarrow{n} b \xrightarrow{r} e$,
- $c \xrightarrow{n} e \xrightarrow{n} b \xrightarrow{r} d \xrightarrow{n} b \xrightarrow{r} d$,
- $c \xrightarrow{n} e \xrightarrow{n} b \xrightarrow{r} d \xrightarrow{n} b \xrightarrow{r} e$,
- $c \xrightarrow{n} e \xrightarrow{n} b \xrightarrow{r} e \xrightarrow{n} b \xrightarrow{r} d$,
- $c \xrightarrow{n} e \xrightarrow{n} b \xrightarrow{r} e \xrightarrow{n} b \xrightarrow{r} e$.

Die Auswertungskantenfolgen können durch die beiden Pfadausdrücke `this.d.b.dref.b.dref` und `this.e.b.dref.b.dref` repräsentiert werden. Der semantisch äquivalente Constraint nach Abflachung der Pfadausdrücke lautet somit:

```
[ some this.d.b.dref.b.dref ++ this.e.b.dref.b.dref ]
```

Als Nächstes zeigen wir die Transformation des folgenden Ausschnitts der Beispiel-Clafer-Spezifikation, die in Abbildung 5.2a gezeigt ist:

```
c 0..*
  f 0..1
    [ some this.parent.g.e ]
  abstract g 0..*
    e 0..*
```

Die Auswertungskantenfolge des Pfadausdrucks, der in dem Constraint enthalten ist, ergibt sich im ursprünglichen unmodifizierten Abhängigkeitsgraphen wie folgt:

$$f \xrightarrow{p} c \xrightarrow{n} g \xrightarrow{n} e.$$

Durch den Pfad wird demnach die abstrakte Clafer-Definition `g` traversiert, für die jedoch keine konkrete Sub-Clafer-Definition existiert. Der Pfadausdruck wird daher durch Hinzunahme der Hilfs-Clafer-Definition `emptyset` wie folgt transformiert:

```
f 0..1
  [ some emptyset ]
emptyset 0..0
```

5.1.4 *Abflachen von Pfadausdrücken in numerischen Ausdrücken*

Als Nächstes betrachten wir die Transformation von Pfadausdrücken, die Pfade repräsentieren, die abstrakte Clafer-Definitionen im Abhängigkeitsgraphen traversieren, und die als Variablen in numerischen Ausdrücken vorkommen. Die Pfadausdrücke repräsentieren in diesem Fall ganzzahlige oder reellwertige Attribute und referenzieren die primitiven Mengen **`integer`** oder **`real`**. Ein Pfadausdruck, der einen Pfad p repräsentiert wird und abstrakte Clafer-Definitionen $c_s \in C_A$ traversiert, fungieren innerhalb numerischer Ausdrücke als Platzhalter für alle Pfade, die über konkrete Sub-Clafer-Definitionen

traversieren, die von Clafer-Definition c_s erben. Zulässige Ersetzungen eines Pfades p sind Pfadausdrücke der Auswertungskantenfolgen, die nach dem in Abschnitt 5.1.3 gezeigten Vorgehen durch Algorithmus 2 berechnet werden. Ein Constraint muss für jede mögliche Ersetzung gelten. Aus diesem Grund können Pfadausdrücke nicht durch das im vorherigen Abschnitt gezeigte Vorgehen als Vereinigung aller Pfadausdrücke der Auswertungskantenfolgen eines Pfades repräsentiert werden.

Stattdessen hat ein Constraint $\texttt{bExp} \in \langle boolExpr \rangle$, in dem n Pfadausdrücke vorkommen (die Variablen in numerischen Ausdrücken repräsentieren) n mögliche Stellen für Ersetzungen. Wir schreiben im Folgenden $\texttt{bExp}(w_1, \ldots, w_n)$, um einen Constraint zu kennzeichnen, in dem der erste Pfadausdruck durch den Pfadausdruck einer Auswertungskantenfolge w_1 und in dem der n-te Pfadausdruck durch den Pfadausdruck einer Auswertungskantenfolge w_n ersetzt wurden. Für die Mengen von Auswertungskantenfolgen $W_1, \ldots, W_n$, die jeweils die möglichen Ersetzungen der n Pfadausdrücke angeben und durch Algorithmus 2 berechnet werden, ergibt sich folgende abgeflachte Formulierung:

$$\forall w_1 \in W_1, \ldots, w_n \in W_n \ : \ \texttt{bExp}(w_1, \ldots, w_n).$$

Der Constraint muss somit für alle möglichen Ersetzungen gelten. Zur Beseitigung der Allquantifizierung expandieren wir die abgeflachte Formulierung und führen für jede expandierte Kopie einen zusätzlichen Constraint ein. Es ergeben sich durch die Transformation somit $|W_1| \cdot \ldots \cdot |W_n|$ zusätzliche Constraints.

Folgendes Beispiel zeigt die Abflachung von Pfadausdrücken in numerischen Ausdrücken.

Beispiel 5.5 (Abflachen von Pfadausdrücken innerhalb von numerischen Ausdrücken) In diesem Beispiel betrachten wir folgenden Ausschnitt der Beispiel-Clafer-Spezifikation, die in Abbildung 5.2a gezeigt ist:

```
c
  [ this.a.l.dref > this.a.k.dref ]
```

Der Ausschnitt enthält einen Constraint, der zwei Pfadausdrücke innerhalb eines numerischen Ausdrucks zueinander in Beziehung setzt. Die Pfade beider Pfadausdrücke traversieren im Abhängigkeitsgraphen die abstrakte Clafer-Definition `a`. Beide Pfadausdrücke müssen daher zur Abflachung der Clafer-Spezifikation transformiert werden.

Zunächst betrachten wir den Pfadausdruck `this.a.l.dref`. Dieser wird von Algorithmus 2 zu folgender Menge W_1 von Auswertungskantenfolgen aufgelöst, die beide auf die Hilfs-Clafer-Definition $c_z \in C_{\mathbb{Z}}$ zeigen:

$$W_1 = \{ (c \xrightarrow{n} d \xrightarrow{n} l \xrightarrow{r} c_z), (c \xrightarrow{n} e \xrightarrow{n} l \xrightarrow{r} c_z) \}.$$

Die zurückübersetzten Pfadausdrücke der Auswertungskantenfolgen lauten `this.d.l.dref` und `this.e.l.dref`.

Entsprechend kann der Pfadausdruck `this.a.k.dref` zu folgender Menge von Auswertungskantenfolgen aufgelöst werden, die jeweils auf die Hilfs-Clafer-Definition $c'_z \in C_{\mathbb{Z}}$ zeigen:

$$W_2 = \{ (c \xrightarrow{n} d \xrightarrow{n} k \xrightarrow{r} c'_z), (c \xrightarrow{n} e \xrightarrow{n} k \xrightarrow{r} c'_z) \}$$

Die zurückübersetzten Pfadausdrücke der Auswertungskantenfolgen lauten `this.d.k.dref` und `this.e.k.dref`.

Die semantisch äquivalente Formulierung des ursprünglichen Constraints lautet somit:

$$\forall w_1 \in W_1, w_2 \in W_2 \, : \, w_1 > w_2$$

Zum Beseitigen der Allquantifizierung expandieren wir im letzten Schritt die Formulierung und erhalten folgende Constraints:

```
[ this.d.l.dref > this.d.k.dref ],
[ this.d.l.dref > this.e.k.dref ],
[ this.e.l.dref > this.d.k.dref ],
[ this.e.l.dref > this.e.k.dref ].
```

Nach Durchführung des in den vorherigen Abschnitten beschriebenen Vorgehens können die nun redundanten abstrakten Clafer-Definitionen sowie Clafer-Definitionen, die transitiv in abstrakten Clafer-Definitionen geschachtelt sind, entfernt werden. Wir bezeichnen die resultierenden abgeflachten Referenzrelationen als $\twoheadrightarrow^F$ und kennzeichnen die Menge der abgeflachten Constraints mit Φ^F. Wir erhalten schließlich eine abgeflachte Clafer-Spezifikation, die wie folgt definiert ist:

Definition 5.6 (Abgeflachte Clafer-Spezifikation) Sei $cs \in CS$ eine nach Definition 4.8 definierte Clafer-Spezifikation. Die abgeflachte Clafer-Spezifikation $\overline{cs}$ ist eine Clafer-Spezifikation mit folgender Signatur: $\overline{cs} = (C^F, \langle identifier \rangle, id, \emptyset, c_r, \emptyset, \emptyset, \blacklozenge\!\!\rightarrow^F, \lambda_c^F, \lambda_g^F, \langle setExpr \rangle, \twoheadrightarrow^F, \langle boolExpr \rangle, \Phi^F)$, wobei durch das in den Abschnitten 5.1.1 bis 5.1.4 beschriebene Vorgehen die Vererbungshierarchien $\leftarrow$ und die Menge aller abstrakten Clafer-Definitionen C_A aus cs beseitigt wurden.

Um die Schreibweise prägnant zu halten, lassen wir im Folgenden für abgeflachte Clafer-Spezifikationen das Superskript F weg und nehmen an, dass die Vererbungshierarchie, alle Constraints sowie alle Referenzen abgeflacht sind.

5.2 REPRÄSENTATION DER SCHACHTELUNGSHIERARCHIE

Im Folgenden beschreiben wir die Transformation T^{ms} einer (abgeflachten) Clafer-Spezifikation $\overline{cs}$ zu einer Multimengen-Repräsentationen $ms \in MS$. Dazu betrachten wir zunächst die Transformation der Schachtelungsrelation $\blacklozenge\!\!\rightarrow$, der Multiplizitätsintervalle

λ_c sowie der Gruppenkardinalitätsintervalle λ_g der Clafer-Definitionen $c \in C$. Eine Multimengen-Repräsentation ms besteht aus der (symbolischen) Multimenge M_C, weiteren Multimengen $M_P \in \mathbb{M}_C$ und Bedingungen zwischen den Zählfunktionen der Multimengen, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ formuliert sind. Eine Belegung der Multimenge M_C repräsentiert hierbei eine Clafer-Spezifikationsinstanz. Um die weiteren Ausführungen prägnant zu halten, schreiben wir im Folgenden, soweit die Bedeutung eindeutig ist, „Multimengen“ und „Zählfunktionen“, wenn „symbolische Multimengen“ und „symbolische Zählfunktionen“ gemäß Definition 4.22 gemeint sind.

Gemäß Definition 4.8 ist das Multiplizitätsintervall der Wurzel-Clafer-Definition $c_r \in C$ gleich $\lambda_c(c_r) := (1,1)$. In jeder Clafer-Spezifikationsinstanz muss die Wurzel-Clafer-Definition c_r deshalb genau einmal vorkommen. Für die Zählfunktion von Multimenge M_C gilt somit: $m_C(c_r) = 1$. Die Anzahl von Instanzen der übrigen Clafer-Definitionen $c \in C \setminus \{c_r\}$ in einer Clafer-Spezifikationsinstanz hängt zusätzlich von der Anzahl von Instanzen der in der Schachtelungshierarchie unmittelbar übergeordneten Clafer-Definition $c' \in C$ (d. h. $c' \blacklozenge\!\!\rightarrow c$) ab. Für ein Multiplizitätsintervall $\lambda_c(c) = (l,u)$ muss die Clafer-Definition c mindestens l- und darf höchstens u-mal je übergeordneter Instanz von c' instanziiert werden. Die Zählfunktion $m_C(c)$ hängt somit vom Multiplizitätsintervall und der Zählfunktion der übergeordneten Clafer-Definition $m_C(c')$ ab. Die Anzahl der Instanzen von Clafer-Definition c ist nach unten hin durch die Bedingung

$$l \cdot m_C(c') \leq m_C(c)$$

beschränkt. Falls das Multiplizitätsintervall unbeschränkt ist und die Bedingung $u = *$ gilt, ist die Anzahl der Instanzen von c nach oben hin unbeschränkt. Ansonsten ist die Anzahl der Instanzen von Clafer-Definition c nach oben hin durch die Bedingung

$$m_C(c) \leq u \cdot m_C(c')$$

beschränkt.

Beispiel 5.7 (Multimengen-Codierung von Multiplizitätsintervallen) In der in Listing 3.1 gezeigten Clafer-Spezifikation ist die Clafer-Definition `members` in der Clafer-Definition `DisseminationGroup` geschachtelt (siehe 19). Zudem trägt die Clafer-Definition `members` das Multiplizitätsintervall 1..*. Es gilt somit $\lambda_c(\texttt{members}) = (1,*)$. Zur Abschätzung der minimalen Anzahl der Instanzen von `members` in einer Clafer-Spezifikationsinstanz führen wir in der Multimengen-Repräsentation folgende Bedingung ein:

$$1 \cdot m_C(\texttt{DisseminationGroup}) \leq m_C(\texttt{members}).$$

Da das Multiplizitätsintervall der Clafer-Definition `members` unbeschränkt ist, wird die maximale Anzahl von Instanzen nicht zusätzlich eingeschränkt.

Des Weiteren hängt die Anzahl der Instanzen einer Clafer-Definition $c \in C \setminus \{c_r\}$ vom Gruppenkardinalitätsintervall $\lambda_g(c') = (l, u)$ der übergeordneten Clafer-Definition $c' \in C$ (d. h. $c' \blacklozenge\!\!\rightarrow c$) ab. Insgesamt muss die Menge der Sub-Clafer-Definitionen $\{c \in C \mid c' \blacklozenge\!\!\rightarrow c\}$ in Summe mindestens l und höchstens u Clafer-Definitions-Instanzen je übergeordneter Instanz der Clafer-Definition c' enthalten. Die Summe der Zählfunktionen aller Sub-Clafer-Definitionen der Multimenge M_C ist somit durch die Bedingung

$$l \cdot m_C(c') \leq \sum_{c \in \{c \in C \mid c' \blacklozenge\!\!\rightarrow c\}} m_C(c)$$

nach unten hin beschränkt[1]. Falls das Gruppenkardinalitätsintervall unbeschränkt ist (d. h. $u = *$ gilt), ist die Summe der Zählfunktionen aller Sub-Clafer-Definitionen nach oben hin unbeschränkt. Ansonsten muss die Bedingung

$$\sum_{c \in \{c \in C \mid c' \blacklozenge\!\!\rightarrow c\}} m_C(c) \leq u \cdot m_C(c')$$

gelten.

Definition 5.8 (Transformation der Schachtelungshierarchie) Zur Transformation eines Multiplizitätsintervalls $\lambda_c(c) = (l_c, u_c)$ und eines Gruppenkardinalitätsintervalls $\lambda_g(c) = (l_g, u_g)$ einer Clafer-Definition $c \in C$ führen wir in der Multimengen-Repräsentation nachfolgende Bedingungen $b_1, b_2 \in \mathcal{L}_{\mathbb{B}}$ ein. Falls λ_c bzw. λ_g unbeschränkt sind (d. h. $u_c = *$ bzw. $u_g = *$ gilt), entfällt die Abschätzung der jeweiligen oberen Schranke.

$$\begin{aligned} b_1 &:= l_c \cdot m_C(c') \leq m_C(c) && \leq u_c \cdot m_C(c') \\ b_2 &:= l_g \cdot m_C(c') \leq \sum_{c \in \{c \in C \mid c' \blacklozenge\!\!\rightarrow c\}} m_C(c) && \leq u_g \cdot m_C(c') \end{aligned}$$

Beispiel 5.9 (Multimengen-Codierung von Gruppenkardinalitätsintervallen) In der in Listing 3.1 gezeigten Spezifikation weist Clafer-Definition `Mode` das Gruppenkardinalitätsintervall 1..1 auf. Die Anzahl der in Clafer-Definition `Mode` enthaltenen Instanzen der Clafer-Definitionen `Emergency` und `Normal` muss daher gleich eins sein. Zum Abschätzen der minimalen Anzahl der Instanzen der geschachtelten Kind-Clafer-Definitionen von `Mode` führen wir daher in der Multimengen-Repräsentation folgende Bedingung ein:

$$1 \cdot m_C(\texttt{Mode}) \leq m_C(\texttt{Emergency}) + m_C(\texttt{Normal}).$$

1 An dieser Stelle sei darauf hingewiesen, dass der verwendete Summenausdruck Teil der Sprache $\mathcal{L}_n$ ist und eine abkürzende Schreibweise für die arithmetische Summe von Zählfunktionsvariablen darstellt.

Entsprechend verwenden wir zum Abschätzen der maximalen Anzahl der geschachtelten Kind-Clafer-Definitionen in der Multimengen-Repräsentation folgende Bedingung:

$$m_C(\texttt{Emergency}) + m_C(\texttt{Normal}) \leq 1 \cdot m_C(\texttt{Mode}).$$

5.3 REPRÄSENTATION DER REFERENZRELATIONEN

Im Folgenden wird die Transformation von Referenzen zwischen Clafer-Definitionen und Mengenausdrücken in die Multimengen-Repräsentation beschrieben. Wenn eine Referenz $c \twoheadrightarrow \texttt{sExp}$ existiert, transformieren wir zunächst den von Clafer-Definition $c \in C$ referenzierten Mengenausdruck $\texttt{sExp} \in \langle setExpr \rangle$ in eine Multimenge M_{Ref} durch folgende Auswertung:

$$M_{Ref} := T_M^{ms}(\texttt{sExp}).$$

Details zur Auswertungsfunktion eines Mengenausdrucks sExp werden in Abschnitt 5.5.1 beschrieben. Um uns auf das Ergebnis dieser Transformation in weiteren Transformationsschritten beziehen zu können, führen wir die partielle Funktion msetRef ein. Die partielle Funktion msetRef ordnet einer Clafer-Definition, welche einen Mengenausdruck referenziert, die Multimenge M_{Ref} zu, die aus der Transformation des Mengenausdrucks resultiert. Da nur ein Teil aller Clafer-Definitionen einer Clafer-Spezifikation eine Referenz aufweist, handelt es sich um nachfolgende partielle Funktion:

$$\text{msetRef} : C \rightharpoonup \mathbb{M}_C.$$

Zur Abschätzung der minimalen und maximalen Anzahl von Instanzen, welche durch die Multimenge M_{Ref} repräsentiert werden, führen wir darüber hinaus folgende Bedingungen $b_1, b_2 \in \mathcal{L}_{\mathbb{B}}$ ein, die für alle Belegungen der Multimengen-Repräsentation erfüllt sein müssen[2]:

(i) $b_1 := \left|M_{Ref}\right| \leq m_C(c) \leq m_C(c') \cdot \left|M_{Ref}\right|$ mit $c' \blacklozenge\!\!\rightarrow c$,

(ii) $b_2 := \forall c \in \text{Supp}(M_{Ref}) : m_{Ref}(c) \leq m_C(c)$.

In Bedingung b_1 gibt die Zählfunktion $m_C(c)$ die Anzahl verschiedener Instanzen an, die in einer Clafer-Spezifikation enthalten sein dürfen. Die Bedingung b_1 stellt somit sicher, dass die durch M_{Ref} repräsentierte Instanzmenge nicht größer ist, als die Anzahl verschiedener Instanzen der Clafer-Definition c. Für eine Clafer-Definition c, die in Clafer-Definition c' geschachtelt ist (d. h. $c' \blacklozenge\!\!\rightarrow c$), gilt, dass k Instanzen von c, die in einer Instanz

2 Der an dieser Stelle erstmalig verwendete Allquantor ist eine abkürzende Schreibweise für die konjunktive Verknüpfung von Ausdrücken der Sprache $\mathcal{L}_{\mathbb{B}}$ der Form $b_2 := b \wedge \ldots \wedge b' \iff m_{Ref}(c) \leq m_C(c) \wedge \ldots \wedge m_{Ref}(c') \leq m_C(c')$ mit $\{c, \ldots, c'\} \subseteq \text{Supp}(M_{Ref})$.

von c' geschachtelt sind, k unterschiedliche Instanzen in `sExp` referenzieren müssen. Die obere Schranke des Zählfunktionswertes $m_C(c)$ kann somit durch den Ausdruck

$$m_C(c') \cdot \left|M_{Ref}\right|$$

abgeschätzt werden. Durch diese Abschätzung entstehen nicht-lineare Abhängigkeiten zwischen (symbolischen) Zählfunktionswerten, die nicht in das lineare mathematische Optimierungsproblem übersetzt werden können, das Zielrepräsentation dient (siehe Transformation T^{ip} in Abschnitt 4.3.3). Zur Abschätzung der Obergrenze des Ausdrucks unterscheiden wir daher zwei Fälle:

(i) Sei Clafer-Definition c' in der Schachtelungshierarchie der Clafer-Definition c übergeordnet (d. h. $c' \blacklozenge\!\!\rightarrow c$). Wenn die Clafer-Definition c' nicht gemäß Definition 4.18 syntaktisch unbeschränkt ist, dann kann der nicht-lineare Ausdruck durch $\lambda_{max}(c') \cdot \left|M_{Ref}\right|$ abgeschätzt werden. Die Funktion λ_{max} ist wie folgt definiert:

$$\begin{aligned} &\lambda_{max}(c_r) := 1, \\ &\lambda_{max}(c) := u \cdot \lambda_{max}(c') \text{ für } \lambda_c(c') := (l, u) \text{ und } c' \blacklozenge\!\!\rightarrow c. \end{aligned}$$

(ii) Wenn die Clafer-Definition c' gemäß Definition 4.18 syntaktisch unbeschränkt ist, kann keine obere Schranke für $m_C(c)$ angegeben werden. In diesem Fall entfällt der nicht-lineare Ausdruck.

Durch Bedingung b_2 wird gefordert, dass die Anzahl der Instanzen einer Clafer-Definition $c'' \in \psi(c)$ (mit $c \twoheadrightarrow$ `sExp`), die durch den Mengenausdruck `sExp` berechnet wird, kleiner oder gleich der Instanzmenge der gleichen Clafer-Definition sein muss, die in der gesamten Clafer-Spezifikation enthalten ist. Die Trägermenge $\text{Supp}(M_{Ref})$ der Multimenge M_{Ref} entspricht hierbei der Menge der Clafer-Definitionen, die durch die statische Auswertungsfunktion $\psi(c)$ berechnet wird. Die Zählfunktionen aller Clafer-Definitionen, die in der Multimenge M_{Ref} enthalten sind, werden daher durch die Bedingung b_2 in Beziehung zur globalen Multimenge M_C gesetzt.

Beispiel 5.10 (Multimengen-Codierung von Referenzen) In der Clafer-Spezifikation, die in Listing 3.1 gezeigt wird, referenziert die Clafer-Definition `members` Instanzen des Mengenausdrucks

```
MobileNode -- AutonomousNode
```

(siehe 19). Die Clafer-Definition `members` ist zudem in der Clafer-Definition `DisseminationGroup` geschachtelt. Durch die in Abschnitt 5.1 beschriebene Abflachung der Vererbungshierarchie wird der Ausdruck umgeformt zu:

```
SlaveNode ++ MasterNode
```

Zunächst wenden wir die Auswertungsfunktion T^{ms} folgendermaßen auf den referenzierten Mengenausdruck an, um eine Multimenge M_{Ref} zu berechnen:

$$M_{Ref} := T^{ms}_{M}(\text{SlaveNode ++ MasterNode}).$$

In der Multimengen-Repräsentation führen wir anschließend folgende zusätzliche Bedingungen zur Transformation der Referenzrelation ein:

(i) $\left|M_{Ref}\right| \leq m_C(\texttt{members}) \leq m_C(\texttt{DisseminationGroup}) \cdot \left|M_{Ref}\right|$
mit $\left|M_{Ref}\right| = m_{Ref}(\texttt{SlaveNode}) + m_{Ref}(\texttt{MasterNode})$,

(ii) $m_{Ref}(\texttt{SlaveNode}) \leq m_C(\texttt{SlaveNode})$ und
$m_{Ref}(\texttt{MasterNode}) \leq m_C(\texttt{MasterNode})$.

Die erste Bedingung stellt sicher, dass die durch M_{Ref} repräsentierte Instanzmenge höchstens so viele Instanzen enthält, wie verschiedene Instanzen der Clafer-Definition `members` existieren, und dass für jede Instanz der Clafer-Definition `members`, die in der gleichen Instanz der Clafer-Definition `DisseminationGroup` geschachtelt ist, jeweils mindestens eine separate Instanz durch M_{Ref} repräsentiert wird.

Die zweite Bedingung stellt sicher, dass die Anzahl der Instanzen je Clafer-Definition `SlaveNode` und `MasterNode` des Mengenausdrucks nicht die Anzahl von Instanzen der jeweiligen Clafer-Definition der gesamten Clafer-Spezifikation M_C übersteigt.

Im Folgenden beschreiben wir die Transformation von Referenzen von Clafer-Definitionen zu den primitiven Mengen **`integer`** und **`real`**. Wenn eine Referenz $c \twoheadrightarrow c_z$ existiert, mit $c \in C$ und $c_z \in C_{\mathbb{Z}}$ bzw. $c_z \in C_{\mathbb{Z}}$, dann ordnen wir einem referenzierten ganzzahligen bzw. reellen Wert eine neue Variable $v \in \mathbb{V}$ folgendermaßen zu:

$$T^{ms}_{n}(c_z) := v.$$

Die Variable v repräsentiert hierbei einen (minimal oder maximal) erlaubten Wert der Instanzmenge der Clafer-Definition c_z. Um auf das Ergebnis dieser Transformation in weiteren Transformationsschritten zurückzugreifen, führen wir folgende Funktion ein, die einer Clafer-Definition, welche die primitive Menge **`integer`** oder **`real`** referenziert, eine Variable $v \in V$ zuordnet:

$$\text{numRef} : C \rightharpoonup V.$$

Definition 5.11 (Transformation von Referenzrelationen) Zur Transformation einer Referenzrelation $c \twoheadrightarrow \texttt{sExp}$ mit $c \in C$ und $\texttt{sExp} \in \langle setExpr \rangle$ und $T^{ms}_{M}(\texttt{sExp}) := M_{Ref}$

führen wir in der Multimengen-Repräsentation folgende zusätzliche Bedingungen $b_1, b_2 \in \mathcal{L}_{\mathbb{B}}$ ein:

$$b_1 := \left|M_{Ref}\right| \leq m_C(c) \leq m_C(c') \cdot \left|M_{Ref}\right| \text{ mit } c' \blacklozenge\!\!\rightarrow c,$$
$$b_2 := \forall c \in \text{Supp}(M_{Ref}) : m_{Ref}(c) \leq m_C(c).$$

Zur Transformation einer Referenzrelation $c \twoheadrightarrow c_z$ mit $c_z \in C_{\mathbb{Z}} \cup C_{\mathbb{R}}$ und $v \in \mathcal{L}_n$ definieren wir die Funktion T_n^{ms} wie folgt:

$$T_n^{ms}(c_z) := v.$$

5.4 REPRÄSENTATION DER PFADAUSDRÜCKE

Besonders herausfordernd bei der Transformation von Clafer-Spezifikationen in die Multimengen-Repräsentation ist der Umgang mit Pfadausdrücken. Im Folgenden betrachten wir, wie Pfadausdrücke `pExpr` $\in \langle pathExpr \rangle$ in die Multimengen-Repräsentation transformiert werden. Pfadausdrücke können entweder innerhalb von Mengenausdrücken `sExp` $\in \langle setExpr \rangle$ oder innerhalb von numerischen Ausdrücken `nExp` $\in \langle numExpr \rangle$ vorkommen (siehe Abschnitt 4.2). Ein Pfadausdruck, der innerhalb eines Mengenausdrucks vorkommt, verweist immer auf eine Menge von Clafer-Definitions-Instanzen. Falls ein Pfadausdruck innerhalb eines numerischen Ausdrucks auftritt, muss dieser immer auf eine Clafer-Definition verweisen, die ein ganzzahliges oder reellwertiges Attribut repräsentiert.

5.4.1 *Kontextabhängigkeit von Pfadausdrücken*

In bestimmten Fällen ist es möglich, kontextabhängige Constraints mit kontextabhängigen Pfadausdrücken in semantisch äquivalente kontextunabhängige Constraints zu übersetzen. Die Kontextabhängigkeit eines Constraints `bExp` $\in \langle boolExpr \rangle$ mit $\Phi(c, \texttt{bExp})$, der in der Kontext-Clafer-Definition c geschachtelt ist, kann nur dann beseitigt werden, wenn für eine Clafer-Definition c und alle Clafer-Definitionen, die in der Schachtelungshierarchie über der Clafer-Definition c liegen, die maximale obere Schranke der Multiplizitätsintervalle gleich eins ist. Die Gültigkeit dieser Bedingung lässt sich wie folgt begründen: Angenommen, dass innerhalb der Schachtelungshierarchie für alle Clafer-Definitionen $c \in C$ nur Multiplizitätsintervalle $\lambda_c(c') = (l, u)$ mit $u \leq 1$ auftreten, dann muss ein in der Clafer-Definition c geschachtelter Constraint `bExp` $\in \langle boolExpr \rangle$ mit $\Phi(c, \texttt{bExp})$ genau dann erfüllt sein, wenn eine Instanz der Clafer-Definition c existiert. Da maximal nur eine Instanz der Clafer-Definition c existieren darf, hängt der Constraint somit nur von höchstens einer Instanz von c ab und kann (bei Vorhandensein einer Instanz von c) global interpretiert werden. Der geschachtelte Constraint `bExp` $\in \langle boolExpr \rangle$ kann somit semantikerhaltend durch die Implikation $c > 0 \implies \texttt{bExp}'$ ersetzt und auf der

obersten Ebene der Schachtelungshierarchie angeordnet werden. Für jeden Pfadausdruck in `bExp'` werden zudem die Schlüsselworte **`this`** und **`parent`** durch ihre absoluten Pfade (ausgehend von der Wurzel-Clafer-Definition $c_r \in C$) ersetzt. Auf diese Weise wird die Kontextabhängigkeit der Pfadausdrücke sowie des Constraints beseitigt.

Beispiel 5.12 (Beseitigung der Kontextabhängigkeit von Pfadausdrücken eines Constraints) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält den Constraint

```
[ some this.Mode <=> some this.mobility ]
```

(siehe 32), der in der Kontext-Clafer-Definition `Context` geschachtelt ist. Der Constraint ist zudem kontextabhängig, da Pfadausdrücke vorkommen, die das lokale Schlüsselwort **`this`** enthalten. Im vorliegenden Beispiel weist die Clafer-Definition `Context` das Multiplizitätsintervall $\lambda_c(\texttt{Context}) = (1,1)$ auf. Der Constraint hängt dementsprechend von höchstens einer Instanz der Clafer-Definition `Context` ab. Wir können somit die Kontextabhängigkeit des Constraints beseitigen, indem wir ihn auf oberster Ebene der Schachtelungshierarchie anordnen und wie folgt umformen:

```
[ Context => (some Context.Mode <=> some Context.mobility) ]
```

Hierbei wurde das Schlüsselwort **`this`** durch den absoluten Pfad zur Kontext-Clafer-Definition `Context` ersetzt und mit dem ursprünglichen Constraint durch eine Implikation verknüpft.

5.4.2 *Schrittweise Transformation von Pfaden*

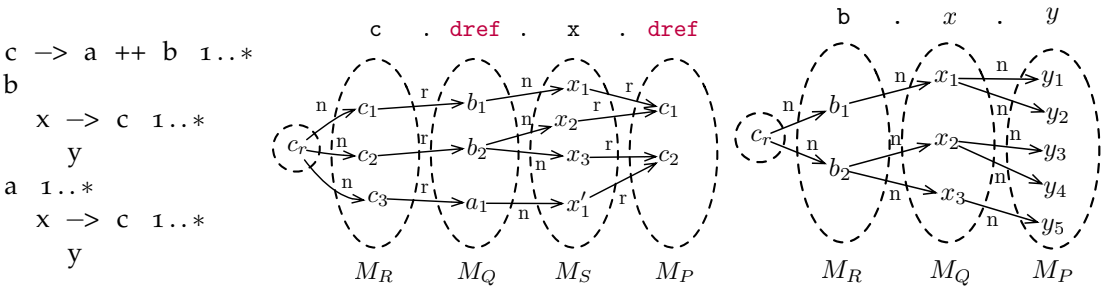
Ein Pfadausdruck kann durch die Verwendung des Schlüsselwortes **`dref`** Instanzen von Clafer-Definitionen potentiell mehrfach referenzieren, soweit diese nicht der gleichen Instanz ihrer Eltern-Clafer-Definition zugeordnet sind.

Beispiel 5.13 (Pfadausdruck mit mehrfach referenzierten Instanzen) Abbildung 5.5a zeigt eine Beispiel-Clafer-Spezifikation, die aus sechs Clafer-Definitionen besteht. Abbildung 5.5b zeigt exemplarisch, wie der Pfadausdruck

```
c.dref.x.dref
```

von der Wurzel-Clafer-Definition c_r ausgehend ausgewertet wird. Im vorliegenden Beispiel werden durch den Pfadausdruck Instanzen der Clafer-Definitionen `c`, `b`, `a`, `x`, `x'`, `y` und `y'` zueinander in Beziehung gesetzt Die Clafer-Definitionen `x` und `x'` bzw. `y` und `y'` entsprechen hierbei unterschiedlichen Clafer-Definitionen mit gleichem Bezeichner.

Innerhalb des Pfadausdrucks gilt für Instanzen der Clafer-Definition `c`, dass diese jeweils verschiedene Instanzen der Clafer-Definitionen `a` und `b` referenzieren müssen, da alle Instanzen von c unter der gleichen Instanz der Clafer-Definition c_r geschach-



(a) Spezifikation **(b)** Allgemeiner Pfadausdruck **(c)** Referenzfreier Pfadausdruck

Abbildung 5.5: Illustration eines allgemeinen und eines referenzfreien Pfadausdrucks

telt sind. Da jeder Instanz der Clafer-Definitionen `x` und `x'` genau eine Eltern-Clafer-Definition `a` bzw. `b` zugeordnet ist, referenzieren innerhalb des Pfadausdrucks Instanzen der Clafer-Definition `a` bzw. `b` ebenfalls verschiedene Instanzen der Clafer-Definition `x` bzw. `x'`.

Instanzen der Clafer-Definitionen `x` und `x'` müssen hingegen nur dann verschiedene Instanzen der Clafer-Definition `c` referenzieren, wenn sie nicht der gleichen Instanz der Eltern-Clafer-Definition untergeordnet sind. Ansonsten können Instanzen der Clafer-Definition `x` bzw. `x'` beliebige Instanzen der Clafer-Definition `c` referenzieren. Instanzen, die durch die transformierte Multimenge $T_M^{ms}(\texttt{c.dref.x.dref}) := M_P$ repräsentiert werden, können somit potentiell mehrfach referenziert werden.

Einen wichtigen Spezialfall für die Abschätzungen der Ober- und Untergrenzen von Multimengen bei der Transformation von Pfadausdrücken stellen Teilpfade von dar, die keine **`dref`**-Schlüsselwörter enthalten und daher verschiedene Instanzen von Clafer-Definitionen referenzieren. Derartige Teilpfade werden im Folgenden als *referenzfreie Teilpfade* bezeichnet. Durch einen referenzfreien Teilpfad kann ausschließlich abwärts (durch den Bezeichner einer Clafer-Definition) oder aufwärts (durch das Schlüsselwort **`parent`**) durch die Schachtelungshierarchie traversiert werden. Insbesondere werden – da das Schlüsselwort **`dref`** nicht auftritt – keine Referenzkanten im Abhängigkeitsgraphen traversiert, die dazu führen, dass Instanzen einer Clafer-Definition potentiell mehrfach referenziert werden.

Die transformierte Multimenge $T_M^{ms}(\texttt{pExpr}) := M_P$ eines referenzfreien Pfadausdrucks repräsentiert somit ausschließlich Instanzen, die genau einmal referenziert werden. Die Anzahl der Instanzen, die durch die Multimenge M_P repräsentiert werden, entspricht in diesem Fall daher der Anzahl verschiedener referenzierter Instanzen eines Pfades.

Beispiel 5.14 (Referenzfreier Pfadausdruck) Abbildung 5.5a zeigt eine Beispiel-Clafer-Spezifikation, die aus sieben Clafer-Definitionen besteht; Abbildung 5.5c zeigt die exemplarische Auswertung des Pfadausdrucks `b.x.y`, der ausgehend von der Wurzel-

Clafer-Definition c_r Instanzen der Clafer-Definition `y` referenziert. Da jeder Instanz einer Kind-Clafer-Definition jeweils genau eine Instanz ihrer Eltern-Clafer-Definition zugeordnet sein muss, können durch den Pfad keine Instanzen der Clafer-Definition `y` mehrfach referenziert werden.

Die transformierte Multimenge $T_M^{ms}(\texttt{b.x.y}) := M_P$ repräsentiert somit die Anzahl verschiedener Instanzen der durch den Pfadausdruck referenzierten Instanzmenge `b.x.y`.

In Definition 5.3 wurde die Auswertungskantenfolge W_i eingeführt, die aus dem Traversieren eines Pfades p im Abhängigkeitsgraphen resultiert. Für einen Pfad p ergeben sich im Abhängigkeitsgraphen eine Menge von Auswertungskantenfolgen W, die bei Auswertung eines Pfades im Abhängigkeitsgraphen schrittweise traversiert werden. Aufgrund von Referenzrelationen, die nach der Abflachung der Schachtelungshierarchie auf mehrere Clafer-Definitionen zeigen, werden in einem Auswertungsschritt potentiell mehrere Kanten des Abhängigkeitsgraphen gleichzeitig traversiert. Basierend auf Definition 5.3 definieren wir nun die Auswertungsschritte eines Pfades wie folgt:

Definition 5.15 (Auswertungsschritte eines Pfades) Seien $DG = (V, E)$ der Abhängigkeitsgraph einer Clafer-Spezifikation und $W = \{ w_1, \ldots, w_n \} = \{ (c_1^1 \xrightarrow{t} c_1^2 \xrightarrow{t} \ldots \xrightarrow{t} c_1^m), \ldots, (c_n^1 \xrightarrow{t} c_n^2 \xrightarrow{t} \ldots \xrightarrow{t} c_n^m) \}$ gemäß Definition 5.3 die Menge der Auswertungskantenfolgen eines abgeflachten Pfades p zu einem Pfadausdruck $\texttt{pExpr} \in \langle pathExpr \rangle$, dann ist ein Auswertungsschritt $W_i^\intercal$ eines Pfades definiert als Kantenmenge $W_i^\intercal = \{ c_1^1 \xrightarrow{t} c_1^2, \ldots, c_n^1 \xrightarrow{t} c_n^2 \}$ mit $W_i^\intercal \subseteq E$. Die Auswertungsschritte eines Pfades sind definiert als Sequenz von Auswertungsschritten $W^\intercal = (W_1^\intercal, \ldots, W_m^\intercal) = (\{ c_1^1 \xrightarrow{t} c_1^2, \ldots, c_n^1 \xrightarrow{t} c_n^2 \}, \ldots, \{ c_1^{m-1} \xrightarrow{t} c_1^m, \ldots, c_n^{m-1} \xrightarrow{t} c_n^m \})$. Der i-te Auswertungsschritt $W_i^\intercal$ eines Pfades entspricht der Menge der i-ten Kanten aller Auswertungskantenfolgen W.

Referenzfreie Teilpfade können wir nun wie folgt charakterisieren:

Definition 5.16 (Referenzfreier Teilpfad) Ein Teilpfad p zu einem Pfadausdruck $\texttt{pExpr} \in \langle pathExpr \rangle$, der durch die Menge der Auswertungsschritte $W^\intercal$ ausgewertet wird, referenziert genau dann verschiedenen Instanzen, wenn alle Auswertungsschritte $W_i^\intercal \in W^\intercal$ jeweils genau eine Kante $c \xrightarrow{n} c'$ oder $c \xrightarrow{p} c'$ enthalten. Insbesondere enthält *kein* Auswertungsschritt eine Referenzkante $c \xrightarrow{r} c'$.

Einen Pfadausdruck $\texttt{pExpr} \in \langle pathExpr \rangle$, der innerhalb eines Mengenausdrucks bzw. innerhalb eines numerischen Ausdrucks vorkommt, transformieren wir in die Multimengen-Repräsentation, indem wir die Funktionen $T_M^{ms}(\texttt{path})(W_i^\intercal)$ bzw. $T_n^{ms}(\texttt{path})(W_i^\intercal)$ rekursiv auf die Elemente des Pfades `path` für Auswertungsschritte $W_i^\intercal$ anwenden. Bei

Anwendung der Funktion $T_M^{ms}(\texttt{path})(W_i^\intercal)$ auf einen Teilpfad `path` und einen Auswertungsschritt $W_i^\intercal$, wird dieser einer Multimenge M_P mit der Trägermenge $\text{Supp}(M_P) := P$ (mit $P \subseteq C$) zugeordnet und zu den Multimengen des restlichen Teilpfads durch die Bedingungen $b_l, b_g \in \mathcal{L}_\mathbb{B}$ in Beziehung gesetzt. Hierbei unterscheiden wir zwischen Bedingungen zur Beschreibung lokaler (b_l) und globaler Abhängigkeiten (b_g). Entsprechend wird die Funktion $T_n^{ms}(\texttt{path})(W_i^\intercal)$ auf einen Pfad `path` und einen Auswertungsschritt $W_i^\intercal$ angewendet, der auf eine Clafer-Definition $c_z \in C_\mathbb{Z}$ bzw. $c_z \in C_\mathbb{R}$ zeigt, die ein ganzzahliges oder reellwertiges Attribut repräsentiert.

Wir definieren nachfolgend die Funktionen T_M^{ms} und T_n^{ms} zur Transformation von Pfadausdrücken in die Multimengen-Repräsentation. Die Abschnitte 5.4.3 bis 5.4.8 enthalten Erläuterungen der einzelnen Transformationsschritte.

Definition 5.17 (Transformation von Pfadausdrücken) Die Funktionen T_M^{ms} und T_n^{ms} zur Transformation eines Pfadausdrucks `pExpr` $\in \langle pathExpr \rangle$ mit Kontext-Clafer-Definition $c_{ctx} \in C$ sind wie folgt definiert:

$$
\begin{array}{lll}
T_M^{ms}(\texttt{this})(W_1^\intercal) & := M_P & \text{mit } W_1^\intercal = \{\, c_{ctx} \xrightarrow{t} c', c_{ctx} \xrightarrow{t} c'', \dots \} \\
 & & \text{Supp}(M_P) = \{\, c_{ctx} \,\} \\
 & & m_C(c_{ctx}) \geq 1 \implies b_l \wedge b_g \\
 & & b_l := m_P(c_{ctx}) = 1 \\
 & & b_g := m_P(c_{ctx}) \leq m_C(c_{ctx}) \\
\hline
T_M^{ms}(\texttt{id})(W_1^\intercal) & := M_P & \text{mit } W_1^\intercal = \{\, c \xrightarrow{t} c', c \xrightarrow{t} c'', \dots \} \\
 & & \text{Supp}(M_P) = \{\, c \,\},\ id(c) = id \\
 & & m_C(c_{ctx}) \geq 1 \implies b_l \wedge b_g \\
 & & b_l := l \leq |M_P| \leq u \text{ mit } \lambda_c(c) = (l, u) \\
 & & b_g := m_P(c) = m_C(c) \\
\hline
T_M^{ms}(\texttt{path.id})(W_i^\intercal) & := M_P & \text{mit } W_i^\intercal = \{\, c_q \xrightarrow{n} c, c_q' \xrightarrow{n} c', \dots \} \text{ und } id(c) = id(c') = id \\
T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) & := M_Q & \text{mit } W_{i-1}^\intercal = \{\, c_s \xrightarrow{t} c_q, c_s' \xrightarrow{t} c_q', \dots \} \\
 & & \text{Supp}(M_P) = \{\, c, c', \dots \},\ \text{Supp}(M_Q) = \{\, c_q, c_q', \dots \} \\
 & & \\
 & & \text{Falls } \texttt{path} \text{ referenzfrei ist:} \\
 & & W_i^\intercal = \{\, c_q \xrightarrow{n} c \,\} \text{ und } \lambda_c(c) = (l, u) \\
 & & m_Q(c_q) \geq 1 \implies b_l \wedge b_g \\
 & & b_l := l \cdot m_Q(c_q) \leq m_P(c) \leq u \cdot m_Q(c_q)
\end{array}
$$

$$b_g := \begin{cases} m_P(c) \leq m_C(c) - (\lambda_{min}(c) - l \cdot m_Q(c_q)) & \text{falls path kontext-abhängig} \\ m_P(c) = m_C(c) & \text{sonst} \end{cases}$$

Sonst:
$\lambda_c(c) = (l, u)$:
$b_l := \left|M_Q\right| \geq 1 \implies l \leq |M_P| \leq u \cdot \left|M_Q\right|$
$b_g := \forall (c_q \xrightarrow{n} c) \in W_i^\intercal : m_Q(c_q) \geq 1 \implies m_P(c) \leq m_C(c)$

$T_M^{ms}(\texttt{path.parent})(W_i^\intercal) := M_P$ mit $W_i^\intercal = \{ c_q \xrightarrow{p} c, c'_q \xrightarrow{p} c', \dots \}$
$T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) := M_Q$ mit $W_{i-1}^\intercal = \{ c_s \xrightarrow{t} c_q, c'_s \xrightarrow{t} c'_q, \dots \}$
$\text{Supp}(M_P) = \{ c, c', \dots \}, \text{Supp}(M_Q) = \{ c_q, c'_q, \dots \}$

Falls path referenzfrei:
$W_i^\intercal = \{ c_q \xrightarrow{p} c \}$ und $\lambda_c(c) = (l, u)$
$m_Q(c_q) \geq 1 \implies b_l \wedge b_g$
$b_l := m_P(c) \leq m_Q(c_q) \leq u \cdot m_P(c)$
$b_g := m_P(c) \leq m_C(c)$

Sonst:
$b_l := \forall (c_q \xrightarrow{p} c) \in W_i^\intercal : m_Q(c_q) \geq 1 \implies 1 \leq m_P(c) \leq m_Q(c_Q)$
$b_g := \forall (c_q \xrightarrow{p} c) \in W_i^\intercal : m_Q(c_q) \geq 1 \implies m_P(c) \leq m_C(c)$

$T_M^{ms}(\texttt{path.dref})(W_i^\intercal) := M_P$ mit $W_i^\intercal = \{ c_q \xrightarrow{r} c, c'_q \xrightarrow{r} c', \dots \}$
$T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) := M_Q$ mit $W_{i-1}^\intercal = \{ c_s \xrightarrow{t} c_q, c'_s \xrightarrow{t} c'_q, \dots \}$
$\text{Supp}(M_P) = \{ c, c', \dots \}, \text{Supp}(M_Q) = \{ c_q, c'_q, \dots \}$

Falls path referenzfrei:
$W_i^\intercal = \{ c_q \xrightarrow{r} c, c_q \xrightarrow{r} c', \dots \}$ mit $c_s \blacklozenge\!\!\rightarrow c_q$
$m_Q(c_q) \geq 1 \implies b_l \wedge b_g$

$$b_l := \begin{cases} m_C(c_q) = |M_P| & \text{falls } c_s = c_r \\ m_C(c_s) \leq |M_P| \leq m_Q(c_q) & \text{sonst} \end{cases}$$

$$b_g := \begin{cases} |M_P| \leq \left|\text{msetRef}(c_q)\right| & \text{falls path kontextabhängig} \\ M_P = M_{Ref} & \text{sonst} \end{cases}$$

mit $M_{Ref} = \text{msetRef}(c_q)$

Sonst:
$b_l := \forall (c_q \xrightarrow{r} c) \in W_i^\intercal : m_Q(c_q) \geq 1 \implies 1 \leq |M_P|$
$b_g := \forall (c_q \xrightarrow{r} c) \in W_i^\intercal : m_Q(c_q) \geq 1 \implies m_P(c) \leq m_{Ref}(c)$
mit $M_{Ref} = \text{msetRef}(c_q)$

$T_n^{ms}(\texttt{path.dref})(W_i^\intercal) := v$ mit $v = \text{numRef}(c_z)$ und $W_i^\intercal = \{ c_q \xrightarrow{r} c_z, c_q' \xrightarrow{r} c_z, \dots \}$
$T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) := M_Q$ mit $W_{i-1}^\intercal = \{ c_s \xrightarrow{t} c_q, c_s' \xrightarrow{t} c_q', \dots \}$

5.4.3 *Teilpfade mit Schlüsselwort this*

Zunächst betrachten wir die Transformation von Teilpfaden gemäß Definition 5.17, die nur aus dem Schlüsselwort **this** bestehen für einen Auswertungsschritt $W_1^\intercal$. Das Schlüsselwort **this** bezieht sich immer auf eine Instanz der Kontext-Clafer-Definition c_{ctx} und steht zu Beginn eines Pfades. Der Auswertungsschritt $W_1^\intercal$ hat somit folgende Form:

$$W_1^\intercal = \{ c_{ctx} \xrightarrow{t} c', c_{ctx} \xrightarrow{t} c'', \dots \}.$$

Über folgende Auswertung der Transformationsfunktion T_M^{ms} wird ein Teilpfad, der aus einem Element mit dem Schlüsselwort **this** besteht, einer Multimenge M_P für einen Auswertungsschritt $W_i^\intercal$ zugeordnet:

$$T_M^{ms}(\textbf{this})(W_1^\intercal) := M_P.$$

Für die Multimenge M_P besteht die Trägermenge aus Instanzen der Clafer-Definition c_{ctx}. Es ergibt sich somit $\text{Supp}(M_P) = \{ c_{ctx} \}$.

Da sich das Schlüsselwort **this** immer genau auf eine Instanz der Clafer-Definition c_{ctx} bezieht, enthält die Multimenge M_P genau eine Instanz. Es ergibt sich somit folgende lokale Bedingung:

$$b_l := m_P(c_{ctx}) = 1.$$

Die gesamte Anzahl von Instanzen der Clafer-Definition c_{ctx} einer Clafer-Spezifikation muss größer sein als die Anzahl der Instanzen der gleichen Clafer-Definition der Multimenge M_P. Es ergibt sich somit folgende globale Bedingung:

$$b_g := m_P(c_{ctx}) \leq m_C(c_{ctx}).$$

Für die Bedingungen b_l und b_g gilt, dass diese erfüllt sein müssen, wenn mindestens eine Instanz der Kontext-Clafer-Definition c_{ctx} in einer Clafer-Spezifikation existiert ($m_C(c_{ctx}) > 0$). Es muss somit folgende Bedingung gelten:

$$m_C(c_{ctx}) > 0 \implies b_l \wedge b_g.$$

Falls die Pfadbedingung eines nicht geschachtelten Constraints transformiert wird, gilt $c_{ctx} = c_r$. Die Prämisse kann in diesem Fall entfallen, da $c_r = 1$ immer gilt.

5.4.4 *Teilpfade mit globalem Bezeichner*

Als Nächstes erläutern wir die Transformationen von Teilpfaden gemäß Definition 5.17 für einen Auswertungsschritt $W_1^\intercal$, die genau einen Bezeichner *id* einer Clafer-Definition $c \in C$ mit $\mathrm{id}(c) = id$, enthalten und ansonsten nicht von der Auswertung eines anderen Teilpfades abhängen. Derartige Teilpfade treten aufgrund der linksrekursiven Auswertungsreihenfolge immer zu Beginn eines Pfadausdrucks auf und beziehen sich auf Clafer-Definitionen, die unterhalb der Wurzel-Clafer-Definition c_r geschachtelt sind. Der Auswertungsschritt $W_1^\intercal$ enthält somit Kanten, die ausgehend von einer in der Wurzel-Clafer-Definition c_r geschachtelten Clafer-Definition c, den Abhängigkeitsgraphen traversieren. Entsprechend hat der Auswertungsschritt $W_1^\intercal$ folgende Form:

$$W_1^\intercal = \{\, c \xrightarrow{t} c', c \xrightarrow{t} c'', \dots \}.$$

Über folgende Auswertung der Transformationsfunktion T_M^{ms} wird ein solcher Teilpfad der Multimenge M_P für einen Auswertungsschritt $W_1^\intercal$ zugeordnet:

$$T_M^{ms}(id)(W_1^\intercal) := M_P.$$

Für die Multimenge M_P besteht die Trägermenge aus Instanzen der Clafer-Definition c ($\mathrm{Supp}(M_P) = \{\, c \,\}$). Die Anzahl der Instanzen der Clafer-Definition c in der Multimenge M_P ist abhängig von ihrem Multiplizitätsintervall $\lambda_c(c) = (l, u)$. Die minimale Anzahl von Instanzen der Multimenge entspricht der unteren Schranke des Multiplizitätsintervalls. Falls das Multiplizitätsintervall nicht unbeschränkt ist (d. h. falls $u \neq *$ gilt) entspricht die maximale Anzahl der Instanzen der Multimenge der oberen Schranke des Multiplizitätsintervalls. Ansonsten ist die Größe der Instanzmenge ebenfalls unbeschränkt. Es ergibt sich somit folgende (lokale) Bedingung:

$$b_l := l \leq |M_P| \leq u.$$

Der zweite Teil der Bedingung $|M_P| \leq u$ entfällt, falls das Multiplizitätsintervall unbeschränkt ist. Ein Teilpfad, der mit dem Bezeichner einer Clafer-Definition beginnt, bezieht sich immer auf alle Instanzen der Clafer-Definition. Die Anzahl der Instanzen der Clafer-Definition c der Multimenge M_P entspricht daher der Anzahl der Instanzen der gesamten Clafer-Spezifikation (M_C). Es gilt somit folgende (globale) Bedingung:

$$b_g := m_P(c) = m_C(c).$$

Die zuvor gezeigte lokale Bedingung b_l zur Abschätzung der Instanzmenge entspricht der in Abschnitt 5.2 präsentierten Abschätzung der Multimenge M_C zur Repräsentation der Schachtelungshierarchie. Da für die Clafer-Definition c die durch die Multimengen M_P und M_C repräsentierten Instanzmengen übereinstimmen, kann in diesem Fall die (redundante) lokale Bedingung b_l entfallen. Für die Bedingungen b_l und b_g gilt, dass diese erfüllt sein müssen, wenn mindestens eine Instanz der Kontext-Clafer-Definition c_{ctx} existiert. Es gilt somit folgende Bedingung:

$$m_C(c_{ctx}) > 0 \implies b_l \wedge b_g.$$

5.4.5 *Teilpfade mit geschachteltem Bezeichner*

Im nächsten Schritt betrachten wir die Transformation von Teilpfaden der Form `path.id` gemäß Definition 5.17, die sich aus einem Bezeichner `id` und einem restlichen Teilpfad `path` zusammensetzen. Über folgende Auswertung der Transformationsfunktion T_M^{ms} für einen Auswertungsschritt $W_i^\intercal$ wird ein solcher Teilpfad den Multimengen M_P und M'_P zugeordnet:

$$T_M^{ms}(\texttt{path.id})(W_i^\intercal) := M_P \text{ und } T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) := M_Q.$$

Die Auswertungsschritte $W_i^\intercal$ und $W_{i-1}^\intercal$ haben hierbei folgende Struktur:

$$W_i^\intercal = \{\, c_q \xrightarrow{n} c, c'_q \xrightarrow{n} c', \dots \},$$
$$W_{i-1}^\intercal = \{\, c_s \xrightarrow{t} c_q, c'_s \xrightarrow{t} c'_q, \dots \}.$$

Für Auswertungsschritt $W_i^\intercal$ wird ausgehend von den Clafer-Definitionen $c_q, c'_q, \dots$ durch die Schachtelungshierarchie abwärts zu den Clafer-Definitionen $c, c', \dots$ traversiert, wobei die Bedingung $id(c) = id(c') = id$ gilt. Die Ausgangspunkte $c_q, c'_q, \dots$ ergeben sich aus dem Auswertungsschritt $W_{i-1}^\intercal$. Die Trägermenge der Multimenge M_P besteht aus der Menge der Clafer-Definitionen, zu denen in Auswertungsschritt $W_i^\intercal$ traversiert wird. Entsprechend ergibt sich die Trägermenge der Multimenge M_Q durch die rekursive Auswertung der Transformationsfunktion $T_M^{ms}(\texttt{path})(W_{i-1}^\intercal)$. Für die Trägermengen gelten somit folgende Bedingungen:

$$\text{Supp}(M_P) = \{\, c, c', \dots \},$$
$$\text{Supp}(M_Q) = \{\, c_q, c'_q, \dots \}.$$

Für die Abschätzung der Ober- und Untergrenze der Multimengen M_P und M_Q unterscheiden wir den Fall, dass der Pfad `path` gemäß Definition 5.16 referenzfrei ist (und somit Instanzen der Instanzmenge genau einmal referenziert) und den Fall, dass der Pfad Instanzen einer Instanzmenge potentiell mehrfach referenziert. Falls der Pfad `path` referenzfrei ist, können schärfere Abschätzungen angegeben werden.

Abschätzung der Ober- und Untergrenze bei referenzfreiem Pfad

Zunächst betrachten wir den Fall, dass der Pfad `path` gemäß Definition 5.16 referenzfrei ist. In diesem Fall besteht der Auswertungsschritt $W_i^\intercal$ nur aus einer Kante $W_i^\intercal = \{\, c_q \xrightarrow{n} c \,\}$. Die Anzahl der Instanzen der Clafer-Definition c der Multimenge M_P ist daher sowohl vom Multiplizitätsintervall $\lambda_c(c) = (l, u)$ als auch von der Anzahl der Instanzen der Multimenge M_Q abhängig. Für jede Instanz von M_Q kann eine Clafer-Definition c mindestens l- und höchstens u-mal instanziiert werden. Falls das Multiplizitätsintervall unbeschränkt ist ($u = *$), kann die Clafer-Definition c beliebig oft instanziiert werden. Die minimale und maximale Anzahl von Instanzen einer Clafer-Definition c, die durch

die Multimenge M_P repräsentiert wird, kann somit durch folgende lokale Bedingung abgeschätzt werden:

$$b_l \coloneqq l \cdot m_Q(c_q) \leq m_P(c) \leq u \cdot m_Q(c_q).$$

Die Beschränkung der Obergrenze entfällt, wenn das Multiplizitätsintervall nach oben unbeschränkt ist.

Zusätzlich hängt die Anzahl der Instanzen einer Clafer-Definition c der Multimenge M_P von der gesamten Anzahl der Instanzen der gleichen Clafer-Definition innerhalb einer Clafer-Spezifikation ab. Falls der Pfadausdruck kontextabhängig ist, kann die Multimenge M_P nur maximal so viele Instanzen der Clafer-Definition c enthalten, wie nicht schon notwendigerweise in anderen Zweigen der Schachtelungshierarchie enthalten sein müssen. Die Funktion $\lambda_{min} : C \to \mathbb{Z}$ liefert die (globale) minimale Anzahl von Instanzen einer Clafer-Definition, indem in der Schachtelungshierarchie aufwärts ausgehend von Clafer-Definition c ($c' \blacklozenge\!\!\rightarrow^{+} c$) alle unteren Schranken der Multiplizitätsintervalle multipliziert werden. Die Funktion λ_{min} ist somit wie folgt definiert:

$$\begin{aligned}&\lambda_{min}(c_r) \coloneqq 1,\\ &\lambda_{min}(c) \coloneqq l \cdot \lambda_{min}(c') \text{ für } \lambda_c(c') \coloneqq (l, u) \text{ und } c' \blacklozenge\!\!\rightarrow c.\end{aligned}$$

Mithilfe der Funktion λ_{min} können wir nun die Mindestanzahl von Instanzen, die bereits in anderen Zweigen der Schachtelungshierarchie enthalten sein müssen, durch den Ausdruck $\lambda_{min}(c) - l \cdot m_Q(c_Q)$ abschätzen. Mit folgender Bedingung lässt sich somit die Multimenge M_P zur Multimenge M_C in Beziehung setzen, falls der Pfad `path` *kontextabhängig* ist:

$$b_g \coloneqq m_P(c) \leq m_C(c) - (\lambda_{min}(c) - l \cdot m_Q(c_q)).$$

Falls der Pfad `path` hingegen *kontextunabhängig* ist, entspricht die Anzahl der Instanzen, die durch die Multimenge M_P repräsentiert wird, der gesamten Anzahl der Instanzen einer Clafer-Spezifikation (repräsentiert durch die Multimenge M_C). Es gilt somit in diesem Fall folgende (globale) Bedingung:

$$b_g \coloneqq m_P(c) = m_C(c).$$

Die Bedingungen b_l und b_g müssen erfüllt sein, falls die vorhergehende Multimenge M_Q mindestens eine Instanz repräsentiert. Es gilt somit die Bedingung

$$m_Q(c_q) \geq 1 \implies b_l \wedge b_g.$$

Abschätzung der Ober- und Untergrenze bei nicht referenzfreiem Pfad

Falls der Pfad `path` Instanzen einer Instanzmenge potentiell mehrfach referenziert, besteht ein Auswertungsschritt gemäß Definition 5.16 im Allgemeinen aus mehreren Kanten $(c_q \xrightarrow{n} c) \in W_i^\mathsf{T}$, die ausgehend von den Clafer-Definitionen $c_q \in \mathrm{Supp}(M_Q)$ die Schachtelungshierarchie abwärts zu den Clafer-Definitionen $c \in \mathrm{Supp}(M_P)$ zeigen.

Falls die vorhergehende Multimenge M_Q mindestens eine Instanz repräsentiert, muss die Multimenge M_P mindestens so viele verschiedene Instanzen einer Clafer-Definition c beinhalten, wie durch die untere Schranke l des Multiplizitätsintervalls $\lambda_c(c) = (l, u)$ gefordert wird. Für die Obergrenze der Multimenge M_P gilt die gleiche Abschätzung wie im referenzfreien Fall, da die Clafer-Definition c_q im Extremfall auf verschiedene Instanzen der Clafer-Definition c zeigt. Die minimale und maximale Anzahl von Instanzen, die durch die Multimenge M_P repräsentiert werden, kann somit durch folgende lokale Bedingung abgeschätzt werden:

$$b_l := |M_Q| \geq 1 \implies l \leq |M_P| \leq u \cdot |M_Q| \,.$$

Je Kante $c_q \xrightarrow{n} c$ des Auswertungsschrittes W_i^T, muss, falls die Multimenge M_Q mindestens eine Instanz der Clafer-Definition c_q repräsentiert (d. h. $m_Q(c_Q) \geq 1$), die Anzahl von Instanzen der Clafer-Definition c in der Multimenge M_P kleiner oder gleich der Anzahl der entsprechenden Clafer-Definition innerhalb der gesamten Clafer-Spezifikation sein. Es ergibt sich somit folgende globale Bedingung:

$$b_g := \forall (c_q \xrightarrow{n} c) \in W_i^\mathsf{T} \,:\, m_Q(c_q) \geq 1 \implies m_P(c) \leq m_C(c).$$

5.4.6 *Teilpfade mit dem Schlüsselwort parent*

Als Nächstes erläutern wir gemäß Definition 5.17 die Transformation von Teilpfaden, die sich aus dem Schlüsselwort **`parent`** und einem vorhergehenden Teilpfad `path` zusammensetzen. Über folgende Auswertung der Transformationsfunktion T_M^{ms} wird ein solcher Teilpfad der Multimenge M_P für einen Auswertungsschritt W_i^T und der Multimenge M_Q für einen Auswertungsschritt W_{i-1}^T zugeordnet:

$$T_M^{ms}(\texttt{path.}\textbf{\texttt{parent}})(W_i^\mathsf{T}) := M_P \text{ und } T_M^{ms}(\texttt{path})(W_{i-1}^\mathsf{T}) := M_Q.$$

Die Auswertungsschritte W_i^T und W_{i-1}^T haben folgende Struktur:

$$W_i^\mathsf{T} = \{\, c_q \xrightarrow{p} c, c_q' \xrightarrow{p} c', \dots \},$$
$$W_{i-1}^\mathsf{T} = \{\, c_s \xrightarrow{t} c_q, c_s' \xrightarrow{t} c_q', \dots \}.$$

Für den Auswertungsschritt W_i^T wird ausgehend von den Clafer-Definitionen $c_q, c_q', \dots$ durch die Schachtelungshierarchie aufwärts zu den Clafer-Definitionen $c, c', \dots$ traversiert. Die Trägermenge der Multimenge M_P besteht aus der Menge der Clafer-Definitionen, zu

denen in Auswertungsschritt $W_i^\intercal$ traversiert wird. Entsprechend ergibt sich die Trägermenge der Multimenge M_Q durch die rekursive Auswertung der Transformationsfunktion $T_M^{ms}(\texttt{path})(W_{i-1}^\intercal)$. Für die Trägermengen gilt somit:

$$\begin{aligned}\text{Supp}(M_P) &= \{\, c, c', \dots \},\\ \text{Supp}(M_Q) &= \{\, c_q, c'_q, \dots \}.\end{aligned}$$

Zur Abschätzung der Ober- und Untergrenze der Multimengen M_P und M_Q unterscheiden wir gemäß Definition 5.16 den Fall, dass der Pfad `path` referenzfrei ist und ausschließlich verschiedene Instanzen referenziert, und den Fall, dass potentiell Instanzen mehrfach durch den Pfad referenziert werden.

Abschätzung der Ober- und Untergrenze bei referenzfreiem Pfad

Für den Fall, dass der Pfad `path` referenzfrei ist, besteht der Auswertungsschritt $W_i^\intercal$ aus nur einer Kante mit $W_i^\intercal = \{\, c_q \xrightarrow{p} c \,\}$. Die Anzahl der Instanzen, die durch die Multimenge M_P repräsentiert wird, ist lokal von der Anzahl der Instanzen der Multimenge M_Q abhängig. Im referenzfreien Fall muss jeder Instanz der Multimenge M_Q mindestens eine Instanz der Multimenge M_P zugeordnet sein. Für ein Multiplizitätsintervall $\lambda_c(c) = (l, u)$ dürfen jeder Instanz der Multimenge M_Q höchstens u Instanzen der Multimenge M_P zugeordnet sein. Es gilt somit folgende lokale Bedingung:

$$b_l \coloneqq m_P(c) \leq m_Q(c_q) \leq u \cdot m_P(c).$$

Die gesamte Anzahl von Instanzen der Clafer-Definition c einer Clafer-Spezifikation muss größer sein, als die Anzahl der Instanzen der gleichen Clafer-Definition der Multimenge M_P. Es ergibt sich somit folgende globale Bedingung:

$$b_g \coloneqq m_P(c) \leq m_C(c).$$

Die Bedingungen b_l und b_g müssen erfüllt sein, falls die vorhergehende Multimenge M_Q mindestens eine Instanz repräsentiert. Es gilt somit:

$$m_Q(c_q) \geq 1 \implies b_l \wedge b_g.$$

Abschätzung der Ober- und Untergrenze bei nicht referenzfreiem Pfad

Falls durch den Pfad `path` Instanzen einer Instanzmenge mehrfach referenziert werden können, besteht ein Auswertungsschritt gemäß Definition 5.16 aus den Kanten $(c_q \xrightarrow{p} c) \in W_i^\intercal$, die ausgehend von den Clafer-Definitionen $c_q \in \text{Supp}(M_Q)$ die Schachtelungshierarchie aufwärts zu den Clafer-Definitionen $c \in \text{Supp}(M_P)$ zeigen.

Falls die Multimenge M_Q mindestens eine Instanz repräsentiert, muss für jede Kante $(c_q \xrightarrow{p} c) \in W_i^\intercal$ einer Instanz der Clafer-Definition c mindestens eine Instanz der in der

Schachtelungshierarchie untergeordneten Kind-Clafer-Definition c_q zugeordnet sein. Die Anzahl der Instanzen der Clafer-Definition c darf jedoch nicht die Anzahl der Instanzen der in der Schachtelungshierarchie untergeordneten Clafer-Definition c_q überschreiten. Es gilt somit folgende lokale Bedingung:

$$b_l := \forall (c_q \xrightarrow{p} c) \in W_i^\intercal \,:\, m_Q(c_q) \geq 1 \implies 1 \leq m_P(c) \leq m_Q(c_Q).$$

In Bezug auf die gesamte Clafer-Spezifikation darf für jede Kante $(c_q \xrightarrow{p} c) \in W_i^\intercal$, falls die vorhergehende Multimenge M_Q mindestens eine Instanz der Clafer-Definition c_q aufweist, die Clafer-Definition c nicht mehr Instanzen in der Multimenge M_P repräsentieren, als in der Clafer-Spezifikation enthalten sind. Es gilt somit folgende globale Bedingung:

$$b_g := \forall (c_q \xrightarrow{p} c) \in W_i^\intercal \,:\, m_Q(c_q) \geq 1 \implies m_P(c) \leq m_C(c).$$

5.4.7 *Teilpfade mit Schlüsselwort dref*

Als Nächstes erläutern wir gemäß Definition 5.17 die Transformation von Teilpfaden, die sich aus dem Schlüsselwort **`dref`** und einem vorhergehenden Teilpfad `path` zusammensetzen. Ein solcher Teilpfad kann entweder Instanzen anderer Clafer-Definitionen oder die primitiven Mengen **`integer`** und **`real`** referenzieren.

In diesem Abschnitt betrachten wir die Transformation von Teilpfaden, die einen Mengenausdruck `sExp` $\in \langle setExpr \rangle$ und somit eine Instanzmenge referenzieren. Über folgende Auswertung der Transformationsfunktion T_M^{ms} wird ein solcher Teilpfad den Multimengen M_P und M_Q für die Auswertungsschritte $W_i^\intercal$ bzw. $W_{i-1}^\intercal$ zugeordnet:

$$T_M^{ms}(\texttt{path.}\textbf{\texttt{dref}})(W_i^\intercal) := M_P \text{ und } T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) := M_P'.$$

Für die Auswertungsschritte $W_i^\intercal$ und $W_{i-1}^\intercal$ gilt folgende Struktur:

$$W_i^\intercal = \{\, c_q \xrightarrow{r} c, c_q' \xrightarrow{r} c', \dots \},$$
$$W_{i-1}^\intercal = \{\, c_s \xrightarrow{t} c_q, c_s' \xrightarrow{t} c_q', \dots \}.$$

Der Auswertungsschritt $W_i^\intercal$ enthält alle Kanten, die ausgehend von den Clafer-Definitionen $c_q, c_q', \dots$ die Clafer-Definitionen $c, c', \dots$ referenzieren. Die Trägermengen der Multimengen M_P und M_Q ergeben sich wie folgt:

$$\text{Supp}(M_P) = \{\, c, c', \dots \},$$
$$\text{Supp}(M_Q) = \{\, c_q, c_q', \dots \}.$$

Die Trägermenge der Multimenge M_P entspricht darüber hinaus der Vereinigung der Trägermengen der von den Clafer-Definitionen $c_q, c_q', \dots$ referenzierten Multimengen, die

aus der Transformation der Referenzrelationen resultieren (siehe Abschnitt 5.3). Es gilt somit zusätzlich folgender Zusammenhang:

$$\text{Supp}(M_P) = \bigcup_{c_q \in \text{Supp}(M_Q)} \text{Supp}(\text{msetRef}(c_q)).$$

Zur Abschätzung der Ober- und Untergrenze der Multimenge M_P unterscheiden wir erneut den Fall, in dem der Pfad `path` referenzfrei ist, und den nicht referenzfreien Fall, in dem Instanzen potentiell mehrfach durch den Pfad referenziert werden können.

Abschätzung der Ober- und Untergrenze bei referenzfreiem Pfad

Falls durch den Pfad `path` gemäß Definition 5.16 auf verschiedene Instanzen einer Instanzmenge verwiesen wird, besteht der Auswertungsschritt $W_i^\intercal$ aus Referenzkanten, die von genau einer Clafer-Definition c_q auf die Clafer-Definitionen $c, c', \ldots$ zeigen. Es gilt somit:

$$W_i^\intercal = \{\, c_q \xrightarrow{r} c, c_q \xrightarrow{r} c', \ldots \}.$$

Die Anzahl der Instanzen, die durch die Multimenge M_P repräsentiert wird, ist lokal von der Anzahl der Instanzen der Multimenge M_Q abhängig. Falls die Clafer-Definition c_q in der Wurzel-Clafer-Definition c_r geschachtelt ist, muss jede Instanz der Multimenge M_Q genau eine Instanz der Multimenge M_P referenzieren. Ansonsten muss pro Instanz der Clafer-Definition c_s, in der Clafer-Definition c_q geschachtelt ist (mit $c_s \blacklozenge\!\!\rightarrow c_q$), mindestens eine Instanz der Clafer-Definition $c \in \text{Supp}(M_P)$ referenziert werden. Es gilt somit folgende lokale Bedingung:

$$b_l := \begin{cases} m_C(c_q) = |M_P| & \text{falls } c_s = c_r \\ m_C(c_s) \leq |M_P| \leq m_Q(c_q) & \text{sonst.} \end{cases}$$

Zusätzlich muss die Anzahl der Instanzen, die durch die Multimenge M_P repräsentiert wird, immer kleiner gleich der Anzahl der Instanzen sein, die durch die Multimenge M_{Ref} repräsentiert wird. Die Multimenge $M_{Ref} = \text{msetRef}(c)$ repräsentiert hierbei die von der Clafer-Definition c global referenzierte Instanzmenge (siehe Abschnitt 5.3). Falls der Pfadausdruck kontextunabhängig ist, entspricht die Multimenge M_P der Multimenge M_{Ref}. Es ergibt sich somit folgende globale Bedingung:

$$b_g := \begin{cases} |M_P| \leq \left|\text{msetRef}(c_q)\right| & \text{falls } \texttt{path} \text{ kontextabhängig} \\ M_P = M_{Ref} & \text{sonst.} \end{cases}$$

Die Bedingungen b_l und b_g müssen gelten, falls die vorhergehende Multimenge M_Q mindestens eine Instanz der Clafer-Definition c_q repräsentiert. Es gilt somit folgende Bedingung:

$$m_Q(c_q) \geq 1 \implies b_l \wedge b_g.$$

Abschätzung der Ober- und Untergrenze bei nicht referenzfreiem Pfad

Falls der Pfad `path` Instanzen einer Instanzmenge potentiell mehrfach referenziert, besteht ein Auswertungsschritt gemäß Definition 5.16 aus mehreren Referenzkanten $c_q \xrightarrow{r} c \in W_i^\mathsf{T}$. Verschiedene Instanzen der Clafer-Definition c_q referenzieren mindestens eine Instanz der Clafer-Definition c. Die Obergrenze von M_P ist hingegen nicht eingeschränkt, da Instanzen der Clafer-Definition c von Instanzen der Clafer-Definition c_q potentiell mehrfach referenziert werden können. Die Abschätzung gilt, falls die vorhergehende Multimenge M_Q mindestens eine Instanz der Clafer-Definition c_q aufweist. Die minimale Anzahl der Instanzen der Multimenge M_P kann somit über folgende lokale Bedingung abgeschätzt werden:

$$b_l := \forall (c_q \xrightarrow{r} c) \in W_i^\mathsf{T} : m_Q(c_q) \geq 1 \implies 1 \leq |M_P| \,.$$

Darüber hinaus darf die Multimenge M_P nicht mehr Instanzen einer Clafer-Definition repräsentieren, als diejenigen Mengenausdrücke `sExp` berechnen, die durch die Clafer-Definitionen c_q mit $c_q \twoheadrightarrow$ `sExp` referenziert werden. Die Instanzmengen der referenzierten Mengenausdrücke `sExp` werden durch die Multimengen $M_{Ref} = \text{msetRef}(c_q)$ repräsentiert (siehe Abschnitt 5.3). Es gilt somit folgende globale Bedingung:

$$b_g := \forall (c_q \xrightarrow{r} c) \in W_i^\mathsf{T} : m_Q(c_q) \geq 1 \implies m_P(c) \leq m_{Ref}(c) \text{ mit } M_{Ref} = \text{msetRef}(c_q).$$

Folgendes Beispiel zeigt die Transformation eines Pfadausdrucks, der eine Instanzmenge referenziert.

Beispiel 5.18 (Multimengen-Codierung eines Pfadausdrucks mit Verweis auf Instanzmenge) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Ausschnitt:

```
Reliable 0..*
  sender -> Node 1..1
  receiver -> Node 1..1
  nodes -> InfrastructureNode ++ AutonomousNode ++ MasterNode ++ SlaveNode 0..2
    [ this.parent.sender.dref ++ this.parent.receiver.dref in this.dref ]
```

Der Ausschnitt zeigt einen in der Clafer-Definition `nodes` geschachtelten Constraint mit dem Pfadausdruck `this.parent.sender.dref`, der eine Instanzmenge referenziert und innerhalb des Constraints mit anderen Instanzmengen in Bezug gesetzt ist. Wir betrachten nun die Transformation dieses Pfadausdrucks in die Multimengen-Repräsentation. Die Transformation erfolgt in vier Schritten, die im Abhängigkeitsgraphen in Abbildung 5.6 mit den Markierungen 1 bis 4 versehen sind. Die Transformationsschritte sind nachfolgend gezeigt. Aus Gründen der Übersichtlichkeit verzichten wir auf die Angabe der Prämissen, dass Multimengen eines vorhergehenden Transformationsschrittes mindestens eine Instanz repräsentieren müssen.

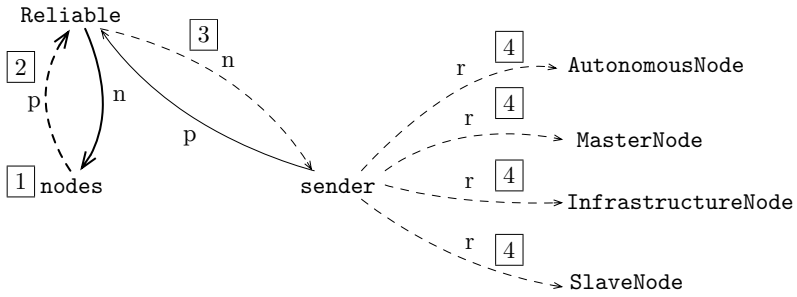


Abbildung 5.6: Beispiel für Auswertungsgraph eines Pfades

1	$T_M^{ms}(\texttt{this}) := M_P$	$\text{Supp}(M_P) = \{\,\texttt{nodes}\,\}$ $b_l := m_P(\texttt{nodes}) = 1$ $b_g := m_P(\texttt{nodes}) \leq m_C(\texttt{nodes})$
2	$T_M^{ms}(\texttt{p.parent}) := M_Q$	$\text{Supp}(M_Q) = \{\,\texttt{Reliable}\,\}$ $b_l := 0 \leq m_Q(\texttt{Reliable})$ $b_g := m_Q(\texttt{Reliable}) \leq m_C(\texttt{Reliable})$
3	$T_M^{ms}(\texttt{p.sender}) := M_R$	$\text{Supp}(M_R) = \{\,\texttt{sender}\,\}$ mit $\lambda_{min}(\texttt{sender}) = 0$ $b_l := m_Q(\texttt{Reliable}) \leq m_R(\texttt{sender}) \leq 1 \cdot m_Q(\texttt{Reliable})$ $\iff m_Q(\texttt{Reliable}) = m_R(\texttt{sender})$ $b_g := m_R(\texttt{sender}) \leq m_C(\texttt{sender}) - (\lambda_{min} - m_Q(\texttt{Reliable}))$ $\iff m_R(\texttt{sender}) \leq m_C(\texttt{sender}) + 1$
4	$T_M^{ms}(\texttt{p.dref}) := M_S$	$\text{Supp}(M_S) = \{\texttt{AutonomousNode}, \texttt{SlaveNode},$ $\texttt{MasterNode}, \texttt{InfrastructureNode}\}$ $b_l := m_C(\texttt{Reliable}) \leq \lvert M_S\rvert \leq m_R(\texttt{sender})$ $\iff m_C(\texttt{Reliable}) \leq \sum_{c \in \text{Supp}(M_S)} m_S(c) \leq m_R(\texttt{sender})$ $b_g := \lvert M_S\rvert \leq \lvert\text{msetRef}(\texttt{sender})\rvert$ $\iff m_S(\texttt{AutonomousNode}) + \ldots + m_S(\texttt{InfrastructureNode}) \leq$ $m_{Ref}(\texttt{AutonomousNode}) + \ldots + m_{Ref}(\texttt{InfrastructureNode})$

Der Pfadausdruck wurde zu den vier Multimengen M_P, M_Q, M_R und M_S transformiert, die durch lokale Bedingungen zueinander in Beziehung gesetzt wurden und durch globale Bedingungen von der Multimenge M_C abhängig sind.

5.4.8 *Teilpfade mit Verweis auf numerische Werte*

Als Nächstes betrachten wir, wie Teilpfade `path.dref`, die numerische Werte referenzieren, gemäß Definition 5.17 in die Multimengen-Repräsentation transformiert werden. In

einem ersten Schritt wird zunächst rekursiv die Transformationsfunktion T_M^{ms} für den Teilpfad `path` ausgewertet, sodass sich Folgendes ergibt:

$$T_M^{ms}(\texttt{path})(W_{i-1}^{\mathsf{T}}) := M_Q.$$

Die Auswertungsschritte W_i^{T} und W_{i-1}^{T} haben folgende Struktur:

$$W_i^{\mathsf{T}} = \{\, c_q \xrightarrow{r} c_z, c_q' \xrightarrow{r} c_z, \dots \},$$
$$W_{i-1}^{\mathsf{T}} = \{\, c_s \xrightarrow{t} c_q, c_s' \xrightarrow{t} c_q', \dots \}.$$

Für den Auswertungsschritt W_i^{T} setzen wir voraus, dass dieser nur Referenzkanten enthält, die auf die gleiche Clafer-Definition $c_z \in C_{\mathbb{Z}}$ bzw. $c_z \in C_{\mathbb{R}}$ zeigen. Auf diese Weise können wir durch die in Abschnitt 5.3 eingeführte Funktion numRef die Referenzen $c_q \twoheadrightarrow c_z$ zu Variable $v := \text{numRef}(c)$ auflösen. Diese Variable $v \in \mathcal{L}_n$ repräsentiert hierbei einen minimalen oder maximalen Wert der durch Clafer-Definition c referenzierten Instanzen der Clafer-Definition c_z. Für die Auswertung der Transformationsfunktion T_n^{ms} ergibt sich somit folgende Zuordnung:

$$T_n^{ms}(\texttt{path.}\textbf{\texttt{dref}})(W_i^{\mathsf{T}}) := \text{numRef}(c_z).$$

Für die gezeigte (globale) Auswertung numerischer Pfadausdrücke sind keine zusätzlichen lokalen Abschätzungen erforderlich.

5.5 REPRÄSENTATION DER CONSTRAINTS

In diesem Abschnitt wird die Transformation von Constraints in die Multimengen-Repräsentation beschrieben. Dazu erläutern wir in Abschnitt 5.5.1, die Transformation von Mengenausdrücken, in Abschnitt 5.5.2 beschreiben wir die Transformation aussagenlogischer Ausdrücke. In den Abschnitten 5.5.3 und 5.5.4 charakterisieren wir die Transformation von Ausdrücken mit einfacher Quantifizierung und Mengenvergleichen. Schließlich beschreiben wir in Abschnitt 5.5.5 die Transformation numerischer Ausdrücke in die Multimengen-Repräsentation.

5.5.1 *Mengenausdrücke*

Mengenausdrücke $\texttt{sExp} \in \langle \textit{setExpr} \rangle$, die in Constraints oder Referenzen vorkommen können, bilden wir in die Multimengen-Repräsentation ab, indem wir neue Multimengen einführen, die zugehörigen Zählfunktionen durch zusätzliche Bedingungen einschränken, und zur Multimenge M_C in Beziehung setzen.

Im Folgenden beschreiben wir die Transformation von Mengenausdrücken $\texttt{sExp} \in \langle \textit{setExpr} \rangle$ in Multimengen sowie Bedingungen zwischen Zählfunktionen. Die Vereinigung zweier Mengenausdrücke ist einer Multimenge $M_R \in \mathbb{M}_C$ wie folgt zugeordnet:

$$T_M^{ms}(\texttt{sExp ++ sExp}') := M_R.$$

Die beiden Mengenausdrücke der Vereinigung sind wiederum rekursiv den Multimengen $M_Q = T_M^{ms}(\texttt{sExp})$ und $M_P = T_M^{ms}(\texttt{sExp}')$ mit $M_Q, M_P \in \mathbb{M}_C$ zugeordnet. Für die aus der Vereinigung resultierende Multimenge M_R lassen sich nun auf Basis der Zählfunktionen $m_Q(c)$ und $m_P(c)$ der zu vereinigenden Multimengen M_Q bzw. M_P Unter- und Obergrenzen der Zählfunktion $m_R(c)$ angeben.

(i) Zur Bestimmung der Untergrenze betrachten wir den Extremfall, dass alle Instanzen einer Clafer-Definition, die durch die Multimenge M_Q repräsentiert werden, auch in der Multimenge M_P enthalten sind oder umgekehrt alle Instanzen, die durch M_P repräsentiert werden, auch in M_Q enthalten sind. In diesem Fall ist die Vereinigungsmenge nur so groß wie die größere der beiden Mengen. Für jede Clafer-Definition $c \in C$ gilt somit folgende Untergrenze für $m_R(c)$:

$$\max(m_Q(c), m_P(c)) \leq m_R(c).$$

(ii) Zur Bestimmung der Obergrenze betrachten wir den Extremfall, dass die Instanzen einer Clafer-Definition, die durch die Multimengen M_Q und M_P repräsentiert werden, vollständig disjunkt sind. In diesem Fall entspricht die Größe der Vereinigungsmenge der Summe der Größen der vereinigten Mengen. Zu beachten ist aber, dass die Vereinigung nicht größer werden kann als die Gesamtmenge aller Instanzen einer Clafer-Definition, die in einer Clafer-Spezifikation enthalten ist (d. h. $m_C(c)$). Für jede Clafer-Definition $c \in C$ gilt somit folgende Obergrenze für $m_R(c)$:

$$m_R(c) \leq \min(m_Q(c) + m_P(c), m_C(c)).$$

Beispiel 5.19 (Multimengen-Codierung der Mengenvereinigung) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält den Ausdruck

```
connections.receiver ++ connections.sender
```

(siehe 12), der zwei Mengenausdrücke vereinigt, die jeweils Instanzen der Clafer-Definition `Node` enthalten. Abbildung 5.7 zeigt schematisch die Abschätzung der Unter- bzw. Obergrenzen der Zählfunktion $m_R(\texttt{AutonomousNode})$ der resultierenden Multimenge M_R. Betrachtet wird die Abschätzung der Zählfunktion für Instanzen der Clafer-Definition `AutonomousNode`, die von der Clafer-Definition `Node` erbt. Im gezeigten Beispiel enthält die Multimenge M_Q drei Instanzen und die Multimenge M_P vier Instanzen der Clafer-Definition `AutonomousNode` (d. h. $m_Q(\texttt{AutonomousNode}) = 3$ und $m_P(\texttt{AutonomousNode}) = 4$). Die Multimenge M_C, welche die Anzahl aller Instanzen einer Spezifikation codiert, enthält sechs Instanzen der Clafer-Definition `AutonomousNode` ($m_C(\texttt{AutonomousNode}) = 6$).

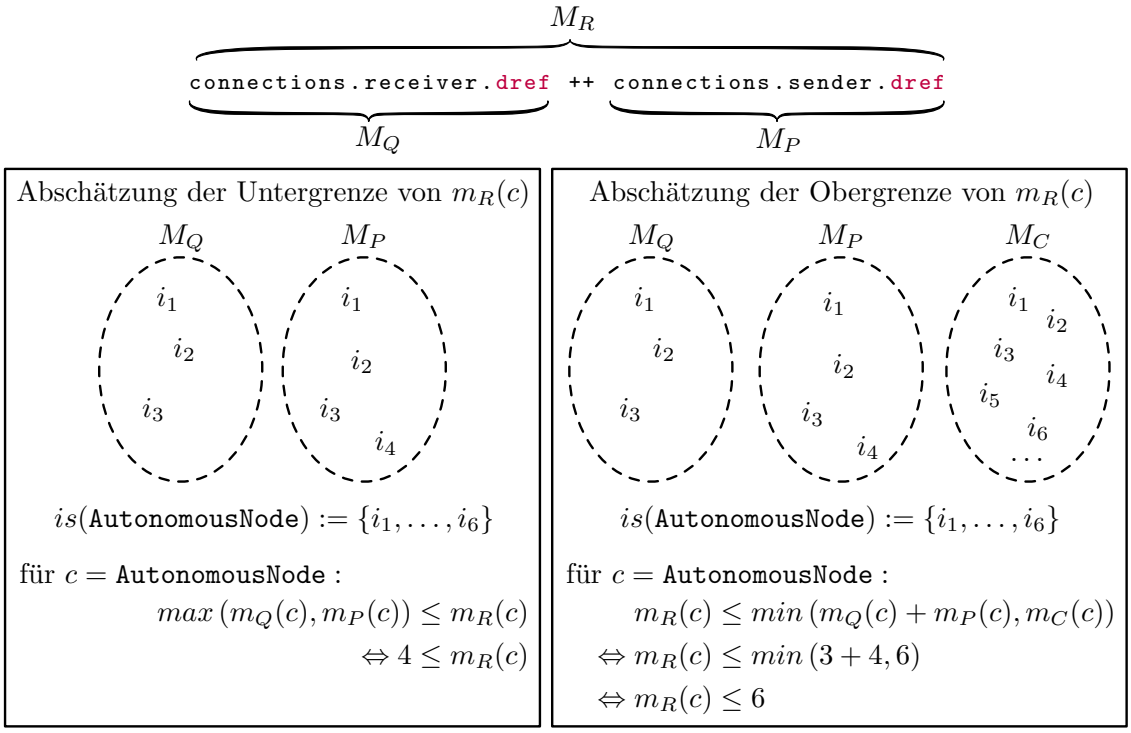


Abbildung 5.7: Beispiel für die Transformation von Vereinigung zweier Mengenausdrücke

Für die Vereinigung der beiden Mengenausdrücke in M_R gilt, dass diese mindestens so viele Instanzen einer Clafer-Definition enthält, wie die maximale Anzahl von Instanzen der beiden einzelnen Mengenausdrücke. Für die Untergrenze von $m_R(\texttt{AutonomousNode})$ gilt somit, dass diese mindestens so groß ist, wie das Maximum von $m_Q(\texttt{AutonomousNode})$ und $m_P(\texttt{AutonomousNode})$, was im gezeigten Beispiel dem Wert vier entspricht.

Die resultierende Multimenge M_R darf hingegen höchstens so viele Elemente einer Clafer-Definition enthalten, wie die Summe der Instanzen der beiden zu vereinigenden Mengenausdrücke. Wenn die Summe jedoch die gesamte Anzahl der Instanzen einer Clafer-Definition einer Clafer-Spezifikation übersteigt, überlappen sich zwangsläufig die beiden zu vereinigenden Mengenausdrücke. In diesem Fall muss die Obergrenze von $m_R(\texttt{AutonomousNode})$ kleiner gleich der Anzahl aller Instanzen der Clafer-Spezifikation sein. Im gezeigten Beispiel kann daher die Obergrenze der Zählfunktion $m_R(\texttt{AutonomousNode})$ mit dem Wert sechs abgeschätzt werden.

Die Differenz zweier Mengenausdrücke kann entsprechend einer Multimenge $M_R \in \mathbb{M}_C$ wie folgt zugeordnet werden:

$$T_M^{ms}(\texttt{sExp -- sExp}') := M_R.$$

Wie zuvor sind die Multimengen $M_Q := T_M^{ms}(\texttt{sExp})$ und $M_P := T_M^{ms}(\texttt{sExp}')$ mit $M_Q, M_P \in \mathbb{M}_C$ den beiden Mengenausdrücken der Differenz zugeordnet. Für die aus der Differenz resultierende Multimenge M_R lassen sich Unter- und Obergrenzen der Zählfunktion $m_R(c)$ angeben.

(i) Zur Bestimmung der Untergrenze betrachten wir den Extremfall, dass alle Instanzen einer Clafer-Definition, die durch die Multimenge M_Q repräsentiert werden, auch in der Multimenge M_P enthalten sind, sodass die resultierende Ergebnismenge leer ist. In diesem Fall entspricht die Anzahl der Instanzen einer Clafer-Definition in der Ergebnismenge der Differenz zwischen der Anzahl der Instanzen der Mengen M_Q und M_P. Falls nicht mindestens so viele Instanzen einer Clafer-Definition in M_P wie in M_Q vorhanden sind, ist die Anzahl in der Ergebnismenge M_R null. Für jede Clafer-Definition $c \in C$ gilt somit folgende Untergrenze für $m_R(c)$:

$$\max(0, m_Q(c) - m_P(c)) \leq m_R(c).$$

(ii) Zur Bestimmung der Obergrenze betrachten wir den Extremfall, dass die Mengen der Instanzen einer Clafer-Definition, die jeweils durch die Multimengen M_Q und M_P repräsentiert werden, vollständig disjunkt sind. In diesem Fall ist die Anzahl von Instanzen einer Clafer-Definition der Multimengen M_R und M_Q gleich. Zu beachten ist dabei aber, dass für eine Clafer-Definition die Multimengen M_Q und M_P nur disjunkt sein können, wenn beide zusammen höchstens so viele Instanzen aufweise wie die Gesamtmenge M_C. Wenn die Summe der Zählfunktionen $m_Q(c)$ und $m_P(c)$ für eine Clafer-Definition $c \in C$ um k größer ist als die Anzahl der Instanzen in M_C, dann müssen M_Q und M_P mindestens k gemeinsame Instanzen von Clafer-Definition c besitzen. Damit ist dann die Zählfunktion $m_R(c)$ um mindestens k Elemente kleiner als $m_Q(c)$, jedoch nicht negativ. Für jede Clafer-Definition $c \in C$ gilt somit folgende Obergrenze für $m_R(c)$:

$$m_R(c) \leq m_Q(c) - \max(0, m_Q(c) + m_P(c) - m_C(c)).$$

Beispiel 5.20 (Multimengen-Codierung der Differenzmenge) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält den Ausdruck

```
MasterNode -- AutonomousNode
```

(siehe 19), der die Differenz zweier Mengenausdrücke bildet, die jeweils Clafer-Definitionen vom Typ `AutonomousNode` enthalten. Abbildung 5.8 zeigt schematisch die Abschätzung der Unter- und Obergrenzen der Zählfunktion $m_R(\texttt{AutonomousNode})$ der resultierenden Multimenge M_R. Betrachtet wird die Abschätzung der Zählfunktion für Instanzen der Clafer-Definition `AutonomousNode`, die von der Clafer-Definition `MobileNode` erbt. Im gezeigten Beispiel enthält die Multimenge M_Q vier

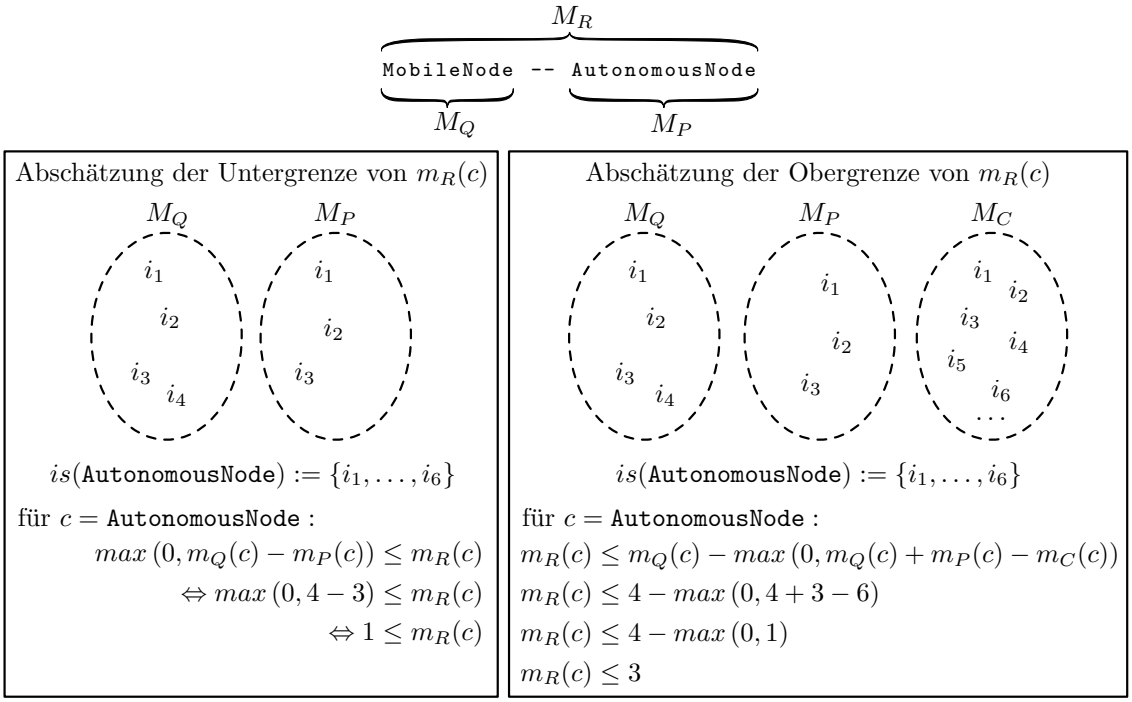


Abbildung 5.8: Beispiel für die Transformation der Differenz zweier Mengenausdrücke

Instanzen und die Multimenge M_P drei Instanzen der Clafer-Definition `AutonomousNode` (d. h. $m_Q(\texttt{AutonomousNode}) = 4$ und $m_P(\texttt{AutonomousNode}) = 3$). Die globale Multimenge M_C enthält sechs Instanzen der Clafer-Definition `AutonomousNode` ($m_C(\texttt{AutonomousNode}) = 6$).

Für die resultierende Multimenge M_R der Differenz der beiden Mengenausdrücke gilt, dass diese mindestens so viele Instanzen einer Clafer-Definition enthält, wie die Differenz der Anzahl von Instanzen der beiden einzelnen Mengenausdrücke. Für die Untergrenze von $m_R(\texttt{AutonomousNode})$ gilt somit, dass diese mindestens so groß ist wie die Differenz von $m_Q(\texttt{AutonomousNode})$ und $m_P(\texttt{AutonomousNode})$. Im gezeigten Beispiel entspricht dies dem Wert eins.

Die resultierende Multimenge M_R darf hingegen höchstens so viele Elemente einer Clafer-Definition enthalten wie die Differenz aus M_Q und der sich zwischen M_Q und M_P überlappenden Instanzen. Im gezeigten Beispiel müssen sich die beiden Mengen M_Q und M_P mit mindestens einer Instanz überlappen ($m_Q(\texttt{AutonomousNode}) + m_P(\texttt{AutonomousNode}) - m_C(\texttt{AutonomousNode}) = 1$). Die Obergrenze der Zählfunktion $m_R(\texttt{AutonomousNode})$ kann daher mit dem Wert drei abgeschätzt werden.

Das Ergebnis einer Schnittmengen-Operation zweier Mengenausdrücke in einer Clafer-Spezifikation kann wie zuvor einer Multimenge $M_R \in \mathbb{M}_C$ zugeordnet werden:

$$T_M^{ms}(\texttt{sExp} \texttt{ ** } \texttt{sExp}') := M_R.$$

Wie bei den bereits gezeigten Mengen-Operationen sind die Multimengen $M_Q := T_M^{ms}(\texttt{sExp})$ und $M_P := T_M^{ms}(\texttt{sExp}')$ mit $M_P, M_Q \in \mathbb{M}_C$ den beiden Mengenausdrücken der Schnittmenge zugeordnet. Für die aus der Schnittmenge resultierende Multimenge M_R lassen sich Unter- und Obergrenze der Zählfunktion $m_R(c)$ angeben.

(i) Zur Bestimmung der Untergrenze betrachten wir den Extremfall, dass die Mengen der Instanzen einer Clafer-Definition $c \in C$, die jeweils durch die Multimengen M_Q und M_P repräsentiert werden, vollständig disjunkt sind. In diesem Fall ist die Ergebnismenge der Schnittmengen-Operation leer, der Zählfunktionswert $m_R(c)$ ist somit gleich null. Wenn allerdings die Anzahl der Instanzen einer Clafer-Definition c in M_Q plus die Anzahl der Instanzen von M_P um k größer ist als die Anzahl aller Instanzen einer Clafer-Spezifikationsinstanz, muss die Schnittmenge M_P mindestens k Instanzen von Clafer-Definition c enthalten. Für jede Clafer-Definition $c \in C$ gilt somit folgende Untergrenze für $m_R(c)$:

$$\max(0, m_Q(c) + m_P(c) - m_C(c)) \leq m_R(c).$$

(ii) Zur Bestimmung der Obergrenze betrachten wir den Extremfall, dass alle Instanzen einer Clafer-Definition, die durch Multimenge M_Q repräsentiert werden, auch in Multimenge M_P enthalten sind oder umgekehrt. In diesem Fall ist die Zählfunktion einer Clafer-Definition $c \in C$ der Multimenge M_R gleich dem Minimum der Zählfunktionen der Multimengen M_Q und M_P. Für jede Clafer-Definition $c \in C$ gilt somit folgende Obergrenze für $m_R(c)$:

$$m_R(c) \leq \min(m_Q(c), m_P(c)).$$

Beispiel 5.21 (Multimengen-Codierung der Schnittmengen-Operation) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält den Ausdruck

```
this.connections.dref.sender.dref ** SlaveNode
```

(siehe ⓯), der die Schnittmenge zweier Mengenausdrücke bildet, die jeweils Instanzen der Clafer-Definitionen `Node` bzw. `SlaveNode` enthalten. Abbildung 5.9 zeigt schematisch die Abschätzung der Unter- bzw. Obergrenze der Zählfunktion $m_R(\texttt{SlaveNode})$ der resultierenden Multimenge M_R. Betrachtet wird die Abschätzung der Zählfunktion für Instanzen der Clafer-Definition `SlaveNode`, die von der Clafer-Definition `MobileNode` erbt. Im gezeigten Beispiel enthält die Multimenge

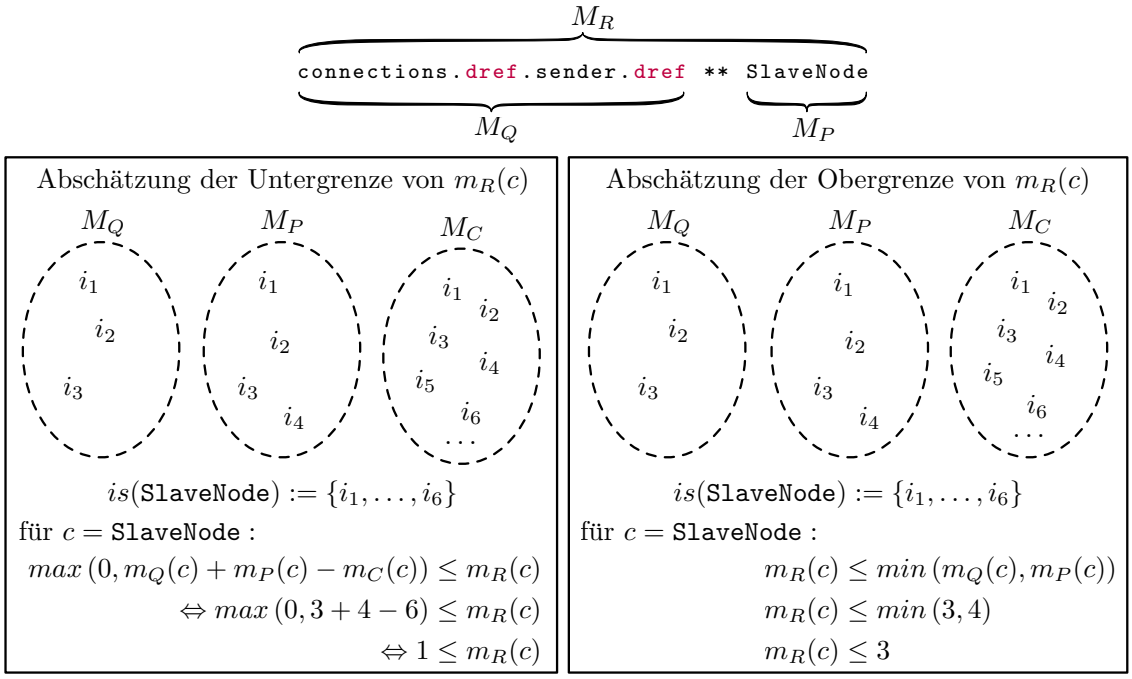


Abbildung 5.9: Beispiel für die Transformation der Schnittmenge zweier Mengenausdrücke

M_Q drei Instanzen und die Multimenge M_P vier Instanzen der Clafer-Definition `SlaveNode` (d. h. $m_Q(\texttt{SlaveNode}) = 3$ und $m_P(\texttt{SlaveNode}) = 4$). Die Multimenge M_C, welche die Anzahl aller Instanzen einer Spezifikation codiert, enthält sechs Instanzen der Clafer-Definition `SlaveNode` ($m_C(\texttt{SlaveNode}) = 6$).

Für die resultierende Multimenge M_R der Schnittmenge der beiden Mengenausdrücke gilt, dass diese mindestens so viele Instanzen einer Clafer-Definition enthält wie sich durch M_Q und M_P maximal überlappen ($m_Q(\texttt{SlaveNode}) + m_P(\texttt{SlaveNode}) - m_C(\texttt{SlaveNode})$). Für die Untergrenze von $m_R(\texttt{SlaveNode})$ gilt somit im gezeigten Beispiel, dass diese mindestens eins ist. Die resultierende Multimenge M_R darf hingegen höchstens so viele Instanzen einer Clafer-Definition enthalten wie der Mengenausdruck mit der geringeren Anzahl von Instanzen. Im gezeigten Beispiel gilt $m_Q(\texttt{SlaveNode}) < m_P(\texttt{SlaveNode})$. Die Obergrenzen von $m_R(\texttt{SlaveNode})$ ist somit kleiner als oder gleich drei.

Definition 5.22 (Transformation von Mengenausdrücken) Sei $T_M^{ms}(\texttt{sExp}) := M_Q$, $T_M^{ms}(\texttt{sExp}') := M_P$ und $b \in \mathcal{L}_{\mathbb{B}}$. Die Funktion T_M^{ms} zur Transformation von Mengenausdrücken ist wie folgt definiert:

$$
\begin{array}{lll}
T^{ms}_{M}(\texttt{sExp ** sExp}') := M_R & b := \max(0, m_Q(c) + m_P(c) - m_C(c)) \leq m_R(c) \\
& \qquad\qquad \leq \min(m_Q(c), m_P(c)) \\
\hline
T^{ms}_{M}(\texttt{sExp -- sExp}') := M_R & b := \max(0, m_Q(c) - m_P(c)) \leq m_R(c) \\
& \qquad \leq m_Q(c) - \max(0, m_Q(c) + m_P(c) - m_C(c)) \\
\hline
T^{ms}_{M}(\texttt{sExp ++ sExp}') := M_R & b := \max(m_Q(c), m_P(c)) \leq m_R(c) \\
& \qquad \leq \min(m_Q(c) + m_P(c), m_C(c)).
\end{array}
$$

5.5.2 *Aussagenlogische Ausdrücke*

Im Folgenden beschreiben wir, wie aussagenlogische Ausdrücke in die Multimengen-Repräsentation transformiert werden. In einem ersten Schritt bringen wir aussagenlogische Ausdrücke $\texttt{bExp} \in \langle \mathit{boolExpr} \rangle$ durch die rekursive Anwendung der in Tabelle 5.3 gezeigten Umformungen in die Negationsnormalform. In dieser Normalform treten Negationsoperatoren nur direkt vor atomaren Aussagen auf [8]. Die negierten atomaren Aussagen können anschließend in die Multimengen-Repräsentation transformiert werden. Konjunktionen und Disjunktionen transformieren wir zu aussagenlogischen Formeln der Multimengen-Repräsentation, indem wir deren Elemente zu aussagenlogischen Formeln transformieren und anschließend konjunktiv bzw. disjunktiv verknüpfen. Die Transformation eines aussagenlogischen Ausdrucks $\texttt{bExp} \in \langle \mathit{boolExpr} \rangle$ und eines konjunktiv- bzw. disjunktiv verknüpften aussagenlogischen Ausdrucks $\texttt{bExp}' \in \langle \mathit{boolExpr} \rangle$ zu einer aussagenlogischen Formel in der Multimengen-Repräsentation erfolgt durch wiederholtes Anwenden der Auswertungsfunktion $T^{ms}_{\mathbb{B}}$:

$$
\begin{aligned}
T^{ms}_{\mathbb{B}}(\texttt{bExp \&\& bExp}') &:= T^{ms}_{\mathbb{B}}(\texttt{bExp})_{\mathbb{B}} \wedge T^{ms}_{\mathbb{B}}(\texttt{bExp}'), \\
T^{ms}_{\mathbb{B}}(\texttt{bExp || bExp}') &:= T^{ms}_{\mathbb{B}}(\texttt{bExp})_{\mathbb{B}} \vee T^{ms}_{\mathbb{B}}(\texttt{bExp}').
\end{aligned}
$$

Beispiel 5.23 (Normalisierung aussagenlogischer Ausdrücke) In der in Listing 3.1 gezeigten Clafer-Spezifikation wird der aussagenlogische Ausdruck

```
!(#connections >= 3 && #connections <= 4)
```

(siehe ❹) durch wiederholtes Anwenden der Umformungen aus Tabelle 5.3 wie folgt in die Negationsnormalform gebracht:

Eingabe	Ausgabe
!(! bExp)	bExp
!(bExp || bExp′)	!bExp && !bExp′
!(bExp && bExp′)	!bExp || !bExp′
bExp => bExp′	!bExp || bExp′
bExp <=> bExp′	bExp => bExp′ && bExp′ => bExp
bExp **xor** bExp′	!bExp && bExp′ || bExp && !bExp′
!(**lone** sExp)	#sExp > 1
!(**one** sExp)	**no** sExp || !(**lone** sExp)
!(**some** sExp)	**no** sExp
!(**no** sExp)	**some** sExp
!(sExp **in** sExp′)	sExp **not in** sExp′
!(sExp **not in** sExp′)	sExp **in** sExp′
!(sExp = sExp′)	sExp != sExp′
!(sExp != sExp′)	nExp = sExp′
!(nExp <= nExp′)	nExp > nExp′
!(nExp >= nExp′)	nExp < nExp′
!(nExp = nExp′)	nExp != nExp′
!(nExp != nExp′)	nExp = nExp′

Tabelle 5.3: Verwendete Korrespondenzen zur Normalisierung von aussagenlogischen Ausdrücken

$$
\begin{aligned}
& !(\#\text{connections} >= 3 \;\&\&\; \#\text{connections} <= 4) \\
\iff\; & !(\#\text{connections} >= 3) \;||\; !(\#\text{connections} <= 4) \\
\iff\; & \underbrace{\#\text{connections} < 3}_{\texttt{bExp}} \;||\; \underbrace{\#\text{connections} > 4}_{\texttt{bExp}'}
\end{aligned}
$$

Für die atomaren Aussagen bExp und bExp′ können wir nun in einem weiteren Schritt Abschätzungen in der Multimengen-Repräsentation angeben.

Beispiel 5.24 (Codierung eines aussagenlogischen Ausdrucks in Multimengen-Repräsentation) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält einen Constraint, der sicherstellt, dass falls die Clafer-Definition `Normal` instanziiert ist, auch mindestens eine Instanz der Clafer-Definitionen `ProbabilisticBroadcast` und `Reliable` existieren müssen (siehe 31). Der Constraint befindet sich bereits in der Negationsnormalform. Daher werden zunächst die Teilausdrücke **some** `ProbabilisticBroadcast` und **some** `Reliable` in Bedingungen über Zählfunktionen der Multimengen-Repräsentation übersetzt, indem die Transformation $T^{ms}_{\mathbb{B}}(\cdot)$ rekursiv auf die Teilausdrücke angewendet wird. Die resultierenden Bedingungen über Zählfunktionen werden anschließend konjunktiv zu einer aussagenlogischen Formel in der Multimengen-Repräsentation wie folgt verknüpft:

$$\underbrace{\textbf{some}\ \texttt{ProbabilisticBroadcast}}_{m_C(\texttt{ProbabilisticBroadcast})\geq 1}\ \underbrace{\texttt{\&\&}}_{\wedge}\ \underbrace{\textbf{some}\ \texttt{Reliable}}_{m_C(\texttt{Reliable})\geq 1}$$

Definition 5.25 (Transformation aussagenlogischer Ausdrücke) Die Funktion $T^{ms}_{\mathbb{B}}$ zur Transformation aussagenlogischer Ausdrücke $\texttt{bExp}, \texttt{bExp}' \in \langle \textit{boolExpr} \rangle$ ist wie folgt definiert:

$$T^{ms}_{\mathbb{B}}(\texttt{bExp \&\& bExp}') := T^{ms}_{\mathbb{B}}(\texttt{bExp})_{\mathbb{B}} \wedge T^{ms}_{\mathbb{B}}(\texttt{bExp}'),$$
$$T^{ms}_{\mathbb{B}}(\texttt{bExp || bExp}') := T^{ms}_{\mathbb{B}}(\texttt{bExp})_{\mathbb{B}} \vee T^{ms}_{\mathbb{B}}(\texttt{bExp}').$$

5.5.3 *Ausdrücke mit einfacher Quantifizierung über Mengen*

Ausdrücke mit dem Schlüsselwörtern **lone**, **one**, **some** und **no** werden in der Sprache Clafer innerhalb von Constraints verwendet, um über Instanzmengen zu quantifizieren, die durch einen Mengenausdruck $\texttt{sExp} \in \langle \textit{setExpr} \rangle$ spezifiziert werden. Zum Beispiel wird durch den Ausdruck **some** `sExp` gefordert, dass die Instanzmenge, die durch den Mengenausdruck `sExp` spezifiziert wird, mindestens eine Instanz beinhaltet. Entsprechend wird durch Ausdrücke mit den Schlüsselwörtern **one**, **lone** und **no** gefordert, dass die Instanzmenge über genau eine, höchstens eine bzw. keine Instanz verfügt.

Zur Transformation der Ausdrücke in die Multimengen-Repräsentation wird zunächst der Mengenausdruck `sExp` einer Multimenge $T^{ms}_{M}(\texttt{sExp}) := M_R$ zugeordnet. Die Gesamtanzahl der Instanzen der Multimenge M_R entspricht der Summe der Zählfunktionswerte $m_R(c)$ aller Clafer-Definitionen $c \in \text{Supp}(M_R)$. Für den Ausdruck **some** `sExp` wird gefordert, dass die Multimenge M_R mindestens eine Instanz repräsentiert. Die Multimenge M_R wird somit durch folgende Bedingung eingeschränkt:

$$|M_P| \geq 1.$$

Andere Ausdrücke zur einfachen Quantifizierung über Instanzmengen können entsprechend in Bedingungen über Zählfunktionen der Multimengen-Repräsentation übersetzt werden (siehe Definition 5.26).

Definition 5.26 (Transformation von Ausdrücken mit einfacher Quantifizierung) Die Funktion $T^{ms}_{\mathbb{B}}$ zur Transformation von Ausdrücken mit einfacher Quantifizierung über Mengenausdrücke $\texttt{sExp} \in \langle setExpr \rangle$ ist wie folgt definiert:

$$
\begin{aligned}
T^{ms}_{\mathbb{B}}(\texttt{lone sExp}) &:= |T^{ms}_{M}(\texttt{sExp})| \leq 1, \\
T^{ms}_{\mathbb{B}}(\texttt{one sExp}) &:= |T^{ms}_{M}(\texttt{sExp})| = 1, \\
T^{ms}_{\mathbb{B}}(\texttt{some sExp}) &:= |T^{ms}_{M}(\texttt{sExp})| \geq 1, \\
T^{ms}_{\mathbb{B}}(\texttt{no sExp}) &:= |T^{ms}_{M}(\texttt{sExp})| = 0.
\end{aligned}
$$

Beispiel 5.27 (Multimengen-Codierung eines Ausdrucks mit einfacher Quantifizierung) In der in Listing 3.1 gezeigten Clafer-Spezifikation stellt der Constraint

```
some this.Interface.LTE
```

(siehe 15) sicher, dass mindestens eine Instanz der Clafer-Definition `LTE` pro Instanz der Clafer-Definition `AutonomousNode` vorkommt. Der Constraint ist in der Clafer-Definition `AutonomousNode` geschachtelt und aufgrund des Vorkommens des Schlüsselwortes `this` kontextabhängig. Zur Abschätzung des Einflusses des Constraints auf die Anzahl der durch die Multimenge M_C repräsentierten Instanzen transformieren wir zunächst den Mengenausdruck `sExp` durch Auswertung der Funktion $T^{ms}_{M}(\texttt{sExp}) = M_R$. Für M_R ergibt sich die Trägermenge zu $\text{Supp}(M_R) = \{\texttt{LTE}\}$, da `sExp` nur Elemente von `LTE` enthalten darf. Wir können somit folgende Abschätzung für die Clafer-Definition `LTE` angeben:

$$\sum_{c \in \text{Supp}(M_R)} m_R(c) \geq 1 \text{ wird umgeformt zu } m_R(\texttt{LTE}) \geq 1$$

Durch die gezeigte Umformung erhalten wir eine obere Schranke der Zählfunktion $m_R(c)$.

5.5.4 *Ausdrücke mit Mengenvergleich*

Für die Transformation von Ausdrücken $\texttt{bExp} \in \langle boolExpr \rangle$ zum Vergleich zweier Mengen (z. B. Ausdrücke der Form `sExp in sExp'` mit $\texttt{sExp}, \texttt{sExp}' \in \langle setExpr \rangle$) wandeln wir zunächst die beiden Teil-Mengenausdrücke zu Multimengen um, sodass sich $T^{ms}_{M}(\texttt{sExp}) = M_P$ und $T^{ms}_{M}(\texttt{sExp}') = M_Q$ ergibt. Zur Abschätzung der Gültigkeit des

Ausdrucks anhand der Zählfunktionswerte der Multimengen M_P und M_Q betrachten wir zwei Fälle.

(i) Falls die Instanzmenge, die durch `sExp` spezifiziert wird, mehr Instanzen aufweist als die Instanzmenge, die durch `sExp'` spezifiziert wird, ist `sExp` notwendigerweise nicht in `sExp'` enthalten und der Ausdruck `sExp in sExp'` somit verletzt. Für alle Clafer-Definitionen c, die in M_P enthalten sind (gegeben durch die Trägermenge $\text{Supp}(M_P)$), muss somit folgende Bedingung erfüllt sein:

$$\forall c \in \text{Supp}(M_p) : m_P(c) \leq m_Q(c).$$

(ii) Falls die Instanzmenge von `sExp` weniger Instanzen als `sExp'` aufweist, kann keine Aussage über die Gültigkeit des Ausdrucks anhand der Zählfunktionen getroffen werden, da beide Instanzmengen im Extremfall vollständig disjunkt sein können.

Mengenvergleiche der Form `sExp = sExp'` werden entsprechend durch folgende Zuordnungen in Bedingungen über Zählfunktionen der Multimengen-Repräsentation übersetzt:

$$T^{ms}_{\mathbb{B}}(\texttt{sExp = sExp}') := \forall c \in \text{Supp}(M_P) \cup \text{Supp}(M_Q) : m_P(c) = m_Q(c).$$

Hingegen kann für negierte Ausdrücke der Form `sExp not in sExp` und `sExp != sExp` anhand der Zählfunktionen der beiden Multimengen M_P und M_Q keine nutzbringende Aussage über die Gültigkeit des Ausdrucks getroffen werden, da sich die Instanzmengen von `sExp` und `sExp'` potentiell beliebig überlappen können und somit keine notwendige Bedingung zwischen den Zählfunktionswerten existiert, die immer erfüllt sein müsste.

Definition 5.28 (Transformation von Ausdrücken mit Mengenvergleich) Die Funktion $T^{ms}_{\mathbb{B}}$ zur Transformation von Ausdrücken, welche die Mengen `sExp` und `sExp` mit `sExp`, `sExp'` $\in \langle setExpr \rangle$ vergleichen, ist wie folgt definiert:

$$T^{ms}_{\mathbb{B}}(\texttt{sExp in sExp}') := \forall c \in \text{Supp}(M_p) : m_P(c) \leq m_Q(c),$$
$$T^{ms}_{\mathbb{B}}(\texttt{sExp = sExp}') := \forall c \in \text{Supp}(M_P) \cup \text{Supp}(M_Q) : m_P(c) = m_Q(c).$$

Beispiel 5.29 (Codierung eines Mengenvergleichs in Multimengen-Repräsentation) In der in Listing 3.1 gezeigten Clafer-Spezifikation ist folgender Constraint enthalten (siehe 17):

$$\underbrace{\texttt{this.parent.sender.dref ++ this.parent.receiver.dref}}_{M_P := T^{ms}_M(\texttt{sExp})} \texttt{ in } \underbrace{\texttt{this.dref}}_{M_Q := T^{ms}_M(\texttt{sExp}')}$$

Der Constraint ist in der Clafer-Definition `nodes` geschachtelt, die aufgrund des Multiplizitätsintervalls 0..2 maximal zwei Instanzen der Clafer-Definition `Node` referenziert. Der Constraint stellt sicher, dass sowohl die von Clafer-Definition `sender` und `receiver` referenzierten Instanzen der Clafer-Definition `Node` in der Instanzmenge enthalten sind, die von der Clafer-Definition `nodes` referenziert wird. Zunächst transformieren wird die beiden Mengenausdrücke zu den Multimengen $M_P := T_M^{ms}(\texttt{sExp})$ und $M_Q := T_M^{ms}(\texttt{sExp}')$. Die Multimenge M_P besteht potentiell aus konkreten Clafer-Definitionen, die von der Clafer-Definition `Node` erben. Die Trägermenge der Multimenge M_P ergibt sich somit zu

$$\text{Supp}(M_P) = \{\, \texttt{SlaveNode}, \texttt{MasterNode}, \texttt{AutonomousNode}, \texttt{InfrastructureNode} \,\}.$$

Wir können nun folgende Abschätzung für die Zählfunktionen der Multimengen M_P und M_Q angeben:

$$\begin{aligned}
& T_{\mathbb{B}}^{ms}(\texttt{sExp in sExp}') && \overset{(1)}{\Longleftrightarrow} \\
& \bigwedge_{c \in \text{Supp}(M_P)} m_P(c) \leq m_Q(c) && \overset{(2)}{\Longleftrightarrow} \\
& \qquad m_P(\texttt{SlaveNode}) \leq m_Q(\texttt{SlaveNode}) && \wedge \\
& \qquad m_P(\texttt{MasterNode}) \leq m_Q(\texttt{MasterNode}) && \wedge \\
& \qquad m_P(\texttt{AutonomousNode}) \leq m_Q(\texttt{AutonomousNode}) && \wedge \\
& \qquad m_P(\texttt{InfrastructureNode}) \leq m_Q(\texttt{InfrastructureNode})
\end{aligned}$$

5.5.5 *Numerische Ausdrücke*

Um numerische Ausdrücke $\texttt{nExp} \in \langle \textit{numExpr} \rangle$ in die Multimengen-Repräsentation zu übersetzen, verwenden wir die Transformationsfunktion T_n^{ms}, die numerische Ausdrücke der Sprache Clafer in arithmetische Ausdrücken der Sprache $\mathcal{L}_n$ zuordnet. Die Transformation des Kardinalitätsoperators # erfolgt, durch folgende Zuordnung:

$$T_n^{ms}(\texttt{\#sExp}) := |T_M^{ms}(\texttt{sExp})|.$$

Hierbei wird zunächst der Mengenausdruck $\texttt{sExp} \in \langle \textit{setExpr} \rangle$ durch die Transformationsfunktion T_M^{ms} in eine Multimenge M_P übersetzt und anschließend durch $|M_P|$ die Summe der (symbolischen) Zählfunktionen gebildet (siehe Abschnitt 4.3 zur Definition des Operators $|\cdot|$). Sonstige numerische Ausdrücke transformieren wir in die Multimengen-Repräsentation, indem wir zunächst die enthaltene Teilausdrücke rekursiv arithmetischen Ausdrücken der Sprache $\mathcal{L}_n$ zuordnen und anschließend durch arithmetische Operationen (der Sprache $\mathcal{L}_n$) verknüpfen.

Definition 5.30 (Transformation numerischer Ausdrücke) Die Funktion T_n^{ms} zur Transformation numerischer Ausdrücke, $\texttt{nExp}, \texttt{nExp}' \in \langle numExpr \rangle$ ist wie folgt definiert:

$$\begin{aligned} T_n^{ms}(\texttt{\#sExp}) &:= |T_M^{ms}(\texttt{sExp})|, \\ T_n^{ms}(-\texttt{nExp}) &:= -T_n^{ms}(\texttt{nExp}), \\ T_n^{ms}(\texttt{nExp} + \texttt{nExp}') &:= T_n^{ms}(\texttt{nExp}) + T_n^{ms}(\texttt{nExp}), \\ T_n^{ms}(\texttt{nExp} - \texttt{nExp}') &:= T_n^{ms}(\texttt{nExp}) - T_n^{ms}(\texttt{nExp}), \\ T_n^{ms}(i * \texttt{nExp}) &:= i \cdot T_n^{ms}(\texttt{nExp}) \text{ mit } i \in \langle numeral \rangle. \end{aligned}$$

Ausdrücke mit Vergleichsoperationen übersetzen wir in Bedingungen der Multimengen-Repräsentation, indem wir numerische Ausdrücke, die Bestandteil der Vergleichsoperationen sind, zu arithmetischen Ausdrücken der Sprache $\mathcal{L}_n$ transformieren und diese anschließend durch Vergleichsoperationen der Sprache $\mathcal{L}_{\mathbb{B}}$ verknüpfen.

Definition 5.31 (Transformation numerischer Vergleichsoperationen) Die Funktion $T_{\mathbb{B}}^{ms}$ zur Transformation von Ausdrücken, die Instanzmengen sExp und sExp mit $\texttt{sExp}, \texttt{sExp}' \in \langle setExpr \rangle$ vergleichen, ist wie folgt definiert:

$$\begin{aligned} T_{\mathbb{B}}^{ms}(\texttt{nExp <= nExp}') &:= T_n^{ms}(\texttt{nExp}) \leq T_n^{ms}(\texttt{nExp}'), \\ T_{\mathbb{B}}^{ms}(\texttt{nExp >= nExp}') &:= T_n^{ms}(\texttt{nExp}) \geq T_n^{ms}(\texttt{nExp}'), \\ T_{\mathbb{B}}^{ms}(\texttt{nExp = nExp}') &:= T_n^{ms}(\texttt{nExp}) = T_n^{ms}(\texttt{nExp}'). \end{aligned}$$

Die Transformation von Pfadausdrücken, die auf Clafer-Definitionen verweisen, die numerische Attribute repräsentieren, wurde in Abschnitt 5.4 beschrieben.

Beispiel 5.32 (Multimengen-Codierung eines numerischen Ausdrucks) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält folgenden Ausschnitt:

```
p -> real
  [ 0 <= this.p.dref && this.p.dref <= 1 && this.p.dref != 0.5 ]
```

In dem enthaltenen Constraint wird die Clafer-Definition p, die ein reellwertiges Attribut repräsentiert, auf den Wertebereich $0 \leq \texttt{p} \leq 1$ eingeschränkt sowie der Wert $\texttt{p} = 0.5$ aus dem Wertebereich ausgeschlossen. Wir betrachten nun die Transformation des Teilausdrucks `0 <= this.p.dref` in die Multimengen-Repräsentation. Zunächst wird der Pfadausdruck `this.p.dref` dem in Abschnitt 5.4 gezeigten Vorgehen entsprechend einer Variablen $v \in V$ der Sprache $\mathcal{L}_n$ zugeordnet:

$$T_n^{ms}(\texttt{this.p.dref}) = v.$$

Entsprechend erfolgt die Transformation der Konstanten `0` zu einer 0 Konstanten der Sprache $\mathcal{L}_n$. Die Transformation der Vergleichsoperation erfolgt durch Anwendung der Transformationsfunktion $T^{ms}_{\mathbb{B}}$. Es ergibt sich somit folgende Transformation:

$$T^{ms}_n(0 <= \texttt{this.p.dref}) := T^{ms}_n(0) \leq T^{ms}_n(\texttt{this.p.dref}),$$
$$\iff 0 \leq v.$$

6

AUTOMATISIERTE ANALYSE VON CLAFER-SPEZIFIKATIONEN

In diesem Kapitel betrachten wir die automatisierte Analyse von Clafer-Spezifikationen. Zunächst untersuchen wir, welche Bestandteile von Clafer Gegenstand der Analyse sind, indem wir die Kernsprache charakterisieren, die durch unser Verfahren analysierbar ist. In einem zweiten Schritt beschreiben wir die Formulierung der in Kapitel 5 eingeführten Multimengen-Repräsentation als mathematisches Optimierungsproblem. Diese Zielrepräsentation ermöglicht die automatisierte Analyse von Clafer-Spezifikationen.

6.1 KERNSPRACHE UND SEMANTISCHE EIGENSCHAFTEN

In diesem Abschnitt beschreiben wir die Kernsprache, die durch unser Verfahren analysierbar ist. Zunächst kennzeichnen wir Sprachelemente der Sprache Clafer, die nicht in die Multimengen-Repräsentation transformiert werden können. Anschließend charakterisieren wir die Kernsprache, die durch unser Verfahren analysierbar ist.

Wir unterscheiden anhand der Eigenschaften Vollständigkeit und Korrektheit unseres Analyseverfahren gemäß Abschnitt 4.3.4 drei Arten von Elementen der Sprache Clafer:

(i) *Kernsprache*: Sprachelemente, die wir in die Multimengen-Repräsentation übersetzen können und für die unser Analyseverfahren vollständig und korrekt ist. Clafer-Spezifikationen, die nur diese Art von Sprachelementen aufweisen, sind Teil der Kernsprache.

(ii) *Erweiterte Kernsprache*: Sprachelemente, die wir in die Multimengen-Repräsentation übersetzen können, für die wir jedoch nur Abschätzungen über Auswirkungen auf Instanzmengen treffen können. Clafer-Spezifikationen, die diese Art von Sprachelementen aufweisen, sind Teil der erweiterten Kernsprache. Analyseergebnisse für Clafer-Spezifikationen, die in der erweiterten Kernsprache vorliegen, sind korrekt. Auch Analyseergebnisse für Spezifikationen, die nicht übersetzbare Elemente enthalten, sind korrekt. Sie sind jedoch unvollständig in dem Sinne, dass nicht alle tatsächlich existierenden Anomalien identifiziert werden können.

(iii) Sprachelemente, für die keine gewinnbringenden Abschätzungen vorgenommen werden können oder die nicht übersetzbar sind.

In Abschnitt 6.1.1 kennzeichnen wir zunächst Sprachelemente, die nicht durch unser Verfahren analysierbar sind. Anschließend charakterisieren wir in Abschnitt 6.1.2 die korrekt analysierbare erweiterte Kernsprache und schließlich in Abschnitt 6.1.3 die sowohl korrekt als auch vollständig analysierbare Kernsprache.

6.1.1 *Analysierbarkeit instanzbasierter und numerischer Ausdrücke*

Eine Instanz $is \in [\![cs]\!]$ einer Clafer-Spezifikation $cs \in CS$ wird in der Multimengen-Repräsentation als Belegung der Multimenge M_C repräsentiert. Es wird somit nur die Gesamtzahl aller Instanzen einer Clafer-Definition $c \in C$ über die zugehörigen Zählfunktionen $m_C(c)$ beachtet, ohne dabei explizit potentielle Abhängigkeiten zwischen einzelnen Instanzen $is(c) = \{ i, i', \dots \}$ einer Clafer-Definition zu berücksichtigen. Aus diesem Grund können Ausdrücke, die explizit Eigenschaften von einzelnen Instanzen einer Clafer-Spezifikation charakterisieren, nicht in die Multimengen-Repräsentation transformiert werden und sind somit nicht analysierbar. Folgende Ausdrücke der Sprache Clafer sind von dieser Einschränkung betroffen:

- *Komplexe Quantifizierungsausdrücke* bestehen aus mehreren lokalen Deklarationen $localDecl \in \langle localDecl \rangle$ und einem aussagenlogischen Ausdruck `bExp` $\in \langle boolExpr \rangle$ (siehe Abbildung 4.1). Eine lokale Deklaration besteht jeweils aus Bezeichnern, die Instanzen referenzieren, und einem Mengenausdruck, der die Instanzmenge spezifiziert, über die quantifiziert wird. Der aussagenlogische Ausdruck charakterisiert Eigenschaften, die (i) für alle, (ii) für genau eine, (iii) für mindestens eine oder (iv) für keine Instanz der Instanzmenge gelten dürfen. Zusätzlich kann durch das Schlüsselwort **`disj`** gefordert werden, dass durch Bezeichner referenzierte Instanzen paarweise verschieden sind. Da komplexe Quantifizierungsausdrücke Instanzen einer Clafer-Definition zueinander in Beziehung setzen und per se einzelne Instanzen einer Clafer-Definition referenzieren, können diese nicht in die Multimengen-Repräsentation transformiert werden und sind daher nicht Teil der erweiterten Kernsprache.

- Die *numerischen Operationen auf Instanzmengen* **`max`**, **`min`**, **`product`** und **`sum`** berechnen das Maximum, Minimum, Produkt bzw. die Summe einer Menge von Instanzen einer Clafer-Definition $c_z \in C_{\mathbb{Z}}$ bzw. $c_z \in C_{\mathbb{R}}$. Zur Auswertung dieser numerischen Operationen müssen alle Instanzen der spezifizierten Instanzmenge zueinander in Beziehung gesetzt werden. Für die Ausdrücke **`max`** und **`min`** lassen sich Abschätzungen angeben, falls die berechnete Instanzmenge nach oben bzw. unten eingeschränkt ist. Da in der Sprache Clafer jedoch Einschränkungen der Wertebereiche von Clafer-Definitionen, die Attribute repräsentieren, nur in Form zusätzlicher Constraints spezifiziert werden können, ist eine statische Auswertung derartiger Einschränkungen nicht möglich. Für die Ausdrücke **`product`** und **`sum`** lassen sich nur für den Spezialfall Abschätzungen angeben, falls die referenzierte Instanzmen-

ge aus höchstens einer Instanz besteht. Im Allgemeinen können die genannten numerischen Operationen hingegen nicht in die Multimengen-Repräsentation transformiert werden und sind nicht Teil der erweiterten Kernsprache.

Beispiel 6.1 (Nicht analysierbarer Ausdruck mit komplexer Quantifizierung) Die in Listing 3.1 gezeigte Clafer-Spezifikation enthält einen Constraint (siehe 18), der aus einem komplexen Quantifizierungsausdruck besteht, in dem über Paare `a`, `b` von Instanzen der Clafer-Definition `Connection` quantifiziert wird. Demnach darf kein disjunktes Paar von Instanzen der Clafer-Definition `Connection` existieren, für welches die referenzierten Instanzen der Clafer-Definitionen `receiver` und `sender` identisch sind. Da in der Multimengen-Repräsentation durch die Zählfunktion $m_C(\texttt{Connection})$ nur die Anzahl von Instanzen der Clafer-Definition `Connection` erfasst wird, kann die Einschränkung der Instanzmenge, die durch diesen Ausdruck spezifiziert wird, nicht repräsentiert werden.

Aufgrund der Zielrepräsentation als mathematisches Optimierungsproblem ergeben sich weitere Einschränkungen bei der Transformation numerischer Ausdrücke: Da die Multimengen-Repräsentation zur Identifikation gültiger Belegungen in ein lineares Optimierungsproblem übersetzt wird, können numerische Operationen der Sprache Clafer, die zu nicht-linearen Nebenbedingungen oder Zielfunktionen des Optimierungsproblems führen, nicht transformiert werden. Folgende numerische Operationen sind aus diesem Grund nicht analysierbar:

- *Multiplikationen* sind in der erweiterten Kernsprache auf Ausdrücke in der Form `k * nExpr` mit $k \in \langle \textit{numeral} \rangle$ und `nExp` $\in \langle \textit{numExpr} \rangle$ eingeschränkt, um potentiell nicht-lineare Multiplikationen von Clafer-Definitionen, die numerische Attribute repräsentieren, zu vermeiden.
- *Divisionsoperationen* der Sprache Clafer werden, soweit sie nicht zu einer Multiplikation der oben gezeigten Form umformbar sind, in der erweiterten Kernsprache nicht unterstützt.
- Ausdrücke mit *Modulo-Operator* sind inhärent nicht-linear und daher kein Bestandteil der erweiterten Kernsprache.

Die genannten Sprachelemente sind nicht in die Multimengen-Repräsentation übersetzbar. Constraints einer Clafer-Spezifikation, welche mindestens eines der genannten Sprachelemente enthalten, werden daher für die Analyse ignoriert. Auf diese Weise können jedoch nicht alle in einer Clafer-Spezifikation existierenden Anomalien erkannt werden, da sie potentiell durch ignorierte Constraints oder durch unscharfe Abschätzungen verursacht werden. In diesem Sinne ist unser Analyseverfahren somit nicht vollständig für Clafer-Spezifikationen mit beliebigen Sprachelementen. Unser Analyseverfahren ist hingegen für beliebige Clafer-Spezifikationen korrekt, da alle Anomalien, die für den analysierbaren

Teil einer Spezifikation identifiziert wurden, auch in der nicht-reduzierten Spezifikation auftreten. Sprachelemente, die durch unser Analyseverfahren analysierbar sind, sind Teil der erweiterten Kernsprache, die im nächsten Abschnitt charakterisiert wird.

6.1.2 *Erweiterte Kernsprache*

Im Folgenden charakterisieren wir zunächst die erweiterte Kernsprache und demonstrieren anhand eines Beispiels, warum die Vollständigkeitseigenschaften unseres Analyseverfahrens für Sprachelemente der erweiterten Kernsprache nicht erfüllt sind.

In Kapitel 5 wurden Transformationen von Elementen der Sprache Clafer in die Multimengen-Repräsentation beschrieben. Dazu wurden mithilfe der Transformationsfunktion T_M^{ms} den Referenzrelationen und Mengenausdrücken die Multimengen $M_P \in \mathbb{M}_C$ zugeordnet. Die Ober- und Untergrenzen der Zählfunktionen der Multimengen wurden durch Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ eingeschränkt und zu Zählfunktionswerten der globalen Multimenge M_C in Beziehung gesetzt. Mithilfe der Transformationsfunktion $T_{\mathbb{B}}^{ms}$ wurden aussagenlogische Ausdrücke, Mengenvergleiche und Ausdrücke mit einfacher Quantifizierung in Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ der Multimengen-Repräsentation übersetzt. Schließlich wurden durch die Transformationsfunktion T_n^{ms} numerische Ausdrücke in Formeln $l \in \mathcal{L}_n$ der Multimengen-Repräsentation übersetzt. Alle Elemente der Sprache Clafer, die sich durch die genannten Transformationsfunktionen T_M^{ms}, $T_{\mathbb{B}}^{ms}$ und T_n^{ms} in die Multimengen-Repräsentation übersetzen lassen, sind Teil der erweiterten Kernsprache. Wir definieren die erweiterte Kernsprache somit wie folgt:

Definition 6.2 (Erweiterte Kernsprache) Die *erweiterte Kernsprache* $CS^{k+} \subseteq CS$ umfasst alle Sprachelemente, die gemäß den Definitionen 5.8, 5.11, 5.17, 5.22, 5.25, 5.26, 5.28 und 5.31 durch die Anwendung der Transformationsfunktionen T_M^{ms}, $T_{\mathbb{B}}^{ms}$ und T_n^{ms} in die Multimengen-Repräsentation übersetzt werden können. Insbesondere ist eine Clafer-Spezifikation $cs \in CS$ Teil der erweiterten Kernsprache CS^{k+}, falls sie *keine* Sprachelemente enthält, die in Abschnitt 6.1.1 gekennzeichnet sind.

Clafer-Spezifikationen, die in der erweiterten Kernsprache vorliegen, sind durch unser Analyseverfahren analysierbar. Für Sprachelemente der erweiterten Kernsprache wurden in Kapitel 5 Abschätzungen der jeweiligen Auswirkungen auf die (globale) Anzahl der Instanzen von Clafer-Definitionen einer Clafer-Spezifikation angegeben. Diese Abschätzungen führen jedoch zu Unschärfen bei der Ermittlung der tatsächlichen minimal oder maximal zulässigen Anzahl von Instanzen von Clafer-Definitionen. Nachfolgend zeigen wir anhand eines Beispiels, wie dies die Vollständigkeitsbedingungen unseres Analyseverfahrens verletzt.

Beispiel 6.3 (Analyse einer Clafer-Spezifikation in der erweiterten Kernsprache) Zur Demonstration der Verletzung der Vollständigkeitseigenschaften unseres Analyseverfahrens für die erweiterte Kernsprache betrachten wir die nachfolgende Beispiel-Clafer-Spezifikation. Die Clafer-Spezifikation liegt in der erweiterten Kernsprache vor, da sie keine der in Abschnitt 6.1.1 genannten Sprachelemente enthält und gemäß Definition 6.2 durch Anwendung der Transformationsfunktionen T_M^{ms}, $T_{\mathbb{B}}^{ms}$ und T_n^{ms} in die Multimengen-Repräsentation übersetzt werden kann.

```
a 2..*
  b 1..*
  [ #(this.b) = 2]
```

Nachfolgend zeigen wir die Transformation der Beispiel-Clafer-Spezifikation in die Multimengen-Repräsentation. Anschließend demonstrieren wir, wie die Multimengen-Repräsentation zur Identifikation von Anomalien genutzt werden kann.

Transformation der Multiplizitätsintervalle:

$$2 \cdot m_C(c_r) \leq m_C(a)$$
$$m_C(a) \leq m_C(b)$$

Transformation des Pfadausdrucks `this.b`:

$$\begin{aligned}
T_M^{ms}(\texttt{this})(W_1^{\mathsf{T}}) \quad &:= M_P \quad \text{mit } \mathrm{Supp}(M_P) = \{\, a \,\} \\
&\quad\; b_l := m_P(a) = 1 \\
&\quad\; b_g := m_P(a) \leq m_C(a) \\
T_M^{ms}(\texttt{this.b})(W_2^{\mathsf{T}}) &:= M_Q \quad \text{mit } \mathrm{Supp}(M_Q) = \{\, b \,\} \\
&\quad\; b_l := m_P(a) \leq m_Q(b) \\
&\quad\; b_g := m_Q(b) \leq m_C(b) - (\lambda_{min}(b) - m_P(a)) \\
&\quad\; \iff m_Q(b) \leq m_C(b) - 1 \qquad (1)
\end{aligned}$$

Transformation des Constraints `#(this.b) = 2`:

$$\begin{aligned}
&|M_Q| = 2 \\
\iff\; & m_Q(b) = 2 \overset{(1)}{\implies} m_C(b) \geq 3 \qquad (2)
\end{aligned}$$

Alle Instanzen der betrachteten Clafer-Spezifikation verfügen wegen Gleichung (2) über mindestens drei Instanzen der Clafer-Definition `b`. Es existiert somit eine Anomalie vom Typ *totes Kardinalitätsintervall* für die Clafer-Definition `b` im Intervall $[0, 2]$. Darüber hinaus existiert eine Anomalie vom Typ *Kern-Clafer-Definition* für die Clafer-Definition `b`, da wegen Gleichung (2) in jeder Instanz der Clafer-Spezifikation mindestens eine Instanz der Clafer-Definition `b` existieren muss. Die getroffene

Abschätzung ist korrekt, da tatsächlich keine Instanz der Clafer-Spezifikation existiert, in der weniger als drei Instanzen der Clafer-Definition `b` vorkommen.

Die Abschätzung liefert jedoch nicht alle tatsächlich existierenden Anomalien: Tatsächlich müssen in jeder Instanz der Clafer-Spezifikation mindestens vier Instanzen der Clafer-Definition `b` vorkommen. Aus diesem Grund kann das durch unsere Analyseverfahren identifizierte Intervall, für das keine Instanzen der Clafer-Definition `b` existiert, erweitert werden auf $[0, 3]$. Die Abschätzung der Auswirkung des kontextabhängigen Constraint `#(this.b) = 2` auf die globale Multimenge M_C führt im gezeigten Beispiel zu einer Unschärfe bei der Ermittlung der minimalen Anzahl von Instanzen der Clafer-Definition `b`.

Das Beispiel demonstriert somit, dass unser Analyseverfahren für die betrachtete Clafer-Spezifikation, die in der erweiterten Kernsprache vorliegt, zwar korrekt aber nicht vollständig ist.

6.1.3 *Kernsprache*

Im Folgenden charakterisieren wir die Kernsprache und weisen die Vollständigkeitseigenschaften unseres Analyseverfahrens für Sprachelemente der Kernsprache nach. Zunächst definieren wir die Kernsprache wie folgt:

Definition 6.4 (Kernsprache) Eine Clafer-Spezifikation cs ist Teil der Kernsprache CS^k, falls sie Teil der erweiterten Kernsprache CS^{k+} gemäß Definition 6.2 ist, mit $cs \in CS^k \subseteq CS^{k+} \subseteq CS$. Zusätzlich müssen alle der folgenden Bedingungen erfüllt sein:

- cs enthält nur Pfadausdrücke, die gemäß Definition 4.13 kontextunabhängig sind. Zusätzlich weisen Pfadausdrücke immer die Form `path` oder `path.dref` auf, wobei der Teilpfad `path` referenzfrei gemäß Definition 5.16 ist.
- alle in cs enthaltenen Referenzrelationen $c \twoheadrightarrow$ `sExp` gehen nur von Clafer-Definitionen $c \in C$ aus, deren Eltern-Clafer-Definitionen $c' \blacklozenge\!\!\rightarrow c$ höchstens einmal instanziierbar sind, sodass die Bedingung $\lambda_{max}(c') \leq 1$ gilt.

Unser Ziel ist es nun, die Gültigkeit des folgenden Satzes zu zeigen:

Satz 6.5 (Vollständigkeit des Analyseverfahrens für Kernsprache) Ist eine Clafer-Spezifikation in der Kernsprache inkonsistent, so besitzt die zugehörige Multimengen-Repräsentation keine Lösung. Unser Analyseverfahren ist vollständig für Clafer-Spezifikationen, die Bestandteil der Kernsprache CS^k gemäß Definition 6.4 sind.

Beweisidee

Das Analyseverfahren ist vollständig, falls jede tatsächliche Anomalie oder Inkonsistenz einer Clafer-Spezifikation durch das Verfahren identifiziert werden kann. Wenn für eine modifizierte Clafer-Spezifikation $cs \oplus \alpha \in CS^k$ mit zusätzlichen Constraints α keine Spezifikationsinstanz gefunden werden kann und somit die Bedingung $[\![cs \oplus \alpha]\!] = \emptyset$ gilt, dann darf auch keine gültige Multimengen-Belegung existieren. Zum Nachweis der Vollständigkeit des Analyseverfahrens gehen wir wie folgt vor:

Beweisschritt 1: Zunächst weisen wir nach, dass Clafer-Spezifikationen, die in der Kernsprache vorliegen, nach der Transformation in die Multimengen-Repräsentation immer eine spezielle Gestalt aufweisen, sodass nur *globale* symbolische Zählfunktionswerte $m_C(c)$ in Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ der Multimengen-Repräsentation auftreten können.

Beweisschritt 2: Anschließend geben wir ein Verfahren an, mit dem sich aus einer konkreten Multimengen-Belegung eine Clafer-Spezifikationsinstanz konstruieren lässt, das die zuvor nachgewiesene spezielle Gestalt der Bedingungen zwischen symbolischen Zählfunktionen ausnutzt.

Beweisschritt 3: Schließlich beweisen wir durch Widerspruch die Gültigkeit von Satz 6.5.

Im Folgenden beschreiben wir die einzelnen Schritte des genannten Vorgehens.

Beweisschritt 1: Multimengen-Repräsentation der Kernsprache

Zunächst zeigen wir für alle Elemente der Kernsprache CS^k, dass diese zusätzlich mit den Annahmen aus Definition 6.4 in der Multimengen-Repräsentation immer zu Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ über ausschließlich globale symbolische Zählfunktionswerte $m_C(c)$ der Multimenge M_C führen. Dazu gehen wir wie folgt vor:

(i) Im ersten Schritt zeigen wir, dass kontextunabhängige und referenzfreie Pfadausdrücke, die in Mengenausdrücken $\texttt{sExp} \in \langle setEpxr \rangle$ vorkommen, die durch andere Clafer-Definitionen $c \in C$ referenziert werden, mit $c \twoheadrightarrow \texttt{sExp}$ und $c' \in \psi(c)$ in der Kernsprache immer zu Multimengen mit globalen Zählfunktionswerten $m_C(c')$ transformiert werden.

(ii) Anschließend zeigen wir, dass die referenzierten Mengenausdrücke `sExp` mit $c \twoheadrightarrow$ `sExp` in der Kernsprache immer zu Multimengen übersetzt werden, die globale Zählfunktionswerte aufweisen.

(iii) Im nächsten Schritt zeigen wir, dass auch Pfadausdrücke, die innerhalb von Mengenausdrücken in Constraints vorkommen, in der Kernsprache immer zu Multimengen transformiert werden, die Zählfunktionswerte der globalen Multimenge M_C aufweisen.

(iv) Entsprechend zeigen wir für Mengenvergleiche (siehe Definition 5.28), Ausdrücke mit einfacher Quantifizierung von Mengen (siehe Definition 5.26) und aussagenlogische Ausdrücke (siehe Definition 5.25), dass die aus der Anwendung der Transformationsfunktion $T^{ms}_{\mathbb{B}}$ resultierenden Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$, in der Kernsprache immer zu Aussagen über alle Instanzen von Clafer-Definitionen einer Clafer-Spezifikation führen.

(v) Schließlich zeigen wir für numerische Ausdrücke (siehe Definition 5.31), dass die Anwendung der Zählfunktion T^{ms}_{n} in der Kernsprache immer zu arithmetischen Formeln führt, die globale und kontextunabhängige Variablen zueinander in Beziehung setzen.

PFADAUSDRÜCKE IN REFERENZIERTEN MENGENAUSDRÜCKEN: Wir betrachten zunächst Pfadausdrücke, die in referenzierten Mengenausdrücken $c \twoheadrightarrow \texttt{sExp}$ auftreten und daher keine **dref**-Schlüsselwörter enthalten dürfen. Eine Clafer-Spezifikation, die in der Kernsprache vorliegt, darf gemäß Definition 6.4 nur kontextunabhängige und referenzfreie Pfadausdrücke $\texttt{pExpr} \in \langle \mathit{pathExpr} \rangle$ enthalten. Gemäß Definition 5.16, und da keine **dref**-Schlüsselwörter auftreten dürfen, hat dies zur Konsequenz, dass ein Pfad p, der durch den Pfadausdruck pExpr repräsentiert wird und die Auswertungsschritte $W^{\intercal} = (W^{\intercal}_1, \ldots, W^{\intercal}_m)$ aufweist, für jeden Auswertungsschritt $W^{\intercal}_i$ mit $1 \leq i \leq m$ aus genau einer Kante $W^{\intercal}_i = \{\, c' \xrightarrow{n} c \,\}$ besteht. Für jeden Auswertungsschritt gelten gemäß Definition 5.17 folgende Bedingungen:

$$
\begin{array}{lll}
T^{ms}_{M}(\texttt{id})(W^{\intercal}_1) := M_P & \text{mit } W^{\intercal}_1 = \{\, c \xrightarrow{t} c', c \xrightarrow{t} c'', \ldots \}, \mathrm{Supp}(M_P) = \{\, c \,\} & \\
 & b_g := m_P(c) = m_C(c) & (1) \\
\hline
T^{ms}_{M}(\texttt{path.id})(W^{\intercal}_i) := M_P & \mathrm{Supp}(M_P) = \{\, c, c', \ldots \} & \\
T^{ms}_{M}(\texttt{path})(W^{\intercal}_{i-1}) := M_Q & \mathrm{Supp}(M_Q) = \{\, c_q, c'_q, \ldots \} & \\
 & W^{\intercal}_i = \{\, c_q \xrightarrow{n} c \,\} \text{ und } \lambda_c(c) = (l, u) & \\
 & b_l := l \cdot m_Q(c_q) \leq m_P(c) \leq u \cdot m_Q(c_q) & (2) \\
 & b_g := m_P(c) = m_C(c). & (3)
\end{array}
$$

Da Pfade in der Kernsprache kontextunabhängig sind, müssen die Gleichungen (1) und (3) erfüllt sein. Jeder Pfadausdruck kann somit einem Zählfunktionswert der globalen Multimenge M_C zugeordnet werden. Durch Gleichung (2) ist zudem sichergestellt, dass Instanzen der Clafer-Definitionen entlang des ausgewerteten Pfades existieren. Jeder kontextunabhängige Pfadausdruck, der innerhalb eines referenzierten Mengenausdrucks auftritt und auf die Clafer-Definition $\mathrm{id}(c) = \mathit{id}$ zeigt, kann somit für die Kernsprache immer durch einen globalen Zählfunktionswert $m_C(c)$ in der Multimengen-Repräsentation ersetzt werden.

MENGENAUSDRÜCKE: Ein Mengenausdruck $\texttt{sExp}'' \in \langle setExpr \rangle$ ist gemäß Definition 4.8 entweder ein einzelner Pfadausdruck, der eine Instanzmenge referenziert, oder eine binäre Mengenoperation, welche die Vereinigung, Differenz oder Schnittmenge von zwei Mengenausdrücken $\texttt{sExp}, \texttt{sExp}' \in \langle setExpr \rangle$ berechnet. Eine Mengenoperation wird gemäß Definition 5.22 in eine Multimenge M_R transformiert, indem die Mengenausdrücke sExp und sExp′ durch Anwendung der Funktion T_M^{ms} zu den Multimengen $T_M^{ms}(\texttt{sExp}) := M_P$ und $T_M^{ms}(\texttt{sExp}') := M_Q$ transformiert werden und indem anschließend durch die Bedingungen $b \in \texttt{bExp}$ die Zählfunktionen beider Multimengen in Beziehung zu der Zählfunktion der Multimenge M_R gesetzt werden.

Ohne Beschränkung der Allgemeinheit nehmen wir an, dass die Multimengen M_P und M_Q für Clafer-Definitionen $c \in C$ nur globale Zählfunktionswerte aufweisen, sodass die Bedingungen $\forall c \in \text{Supp}(M_P) : m_P(c) = m_C(c)$ und $\forall c' \in \text{Supp}(M_Q) : m_Q(c) = m_C(c)$ gelten. Wir haben bereits bei der Betrachtung von Pfadausdrücken gezeigt, dass diese Annahme erfüllt ist, falls sExp und sExp′ jeweils nur aus einem referenzfreien Pfadausdruck bestehen. Für die Vereinigungs-, Differenzmengen- und Schnittmengenoperation können nun zwei Fälle auftreten:

1. Fall: Eine Clafer-Definition $c \in C$ ist zwar in der Trägermenge von M_Q, aber nicht in der Trägermenge von M_P enthalten. In diesem Fall gelten die Bedingungen $m_Q(c) = m_C(c)$ und $m_P(c) = 0$.

2. Fall: Eine Clafer-Definition $c \in C$ ist in der reduzierten Multimenge von M_P und M_Q enthalten. In diesem Fall ist die Bedingung $m_P(c) = m_Q(c) = m_C(c)$ erfüllt.

Im Folgenden zeigen wir für die Transformation der Vereinigungs-, Differenz- und Schnittmengenoperationen in eine Multimenge M_R zu jedem der genannten Fälle, dass Zählfunktionswerte der Clafer-Definitionen $c \in C$ der resultierenden Multimenge $m_R(c)$ durch globale Zählfunktionswerte $m_C(c)$ in der Kernsprache ersetzt werden können.

Für die Transformation der Vereinigungsoperation $\texttt{sExp} \texttt{ ++ } \texttt{sExp}'$ ergibt sich in der Kernsprache gemäß Definition 5.22:

- 1. Fall:
$$\begin{aligned} & \max(m_Q(c), m_P(c)) \leq m_R(c) \leq \min(m_Q(c) + m_P(c), m_C(c)) \\ \iff & \max(m_C(c), 0) \leq m_R(c) \leq \min(m_C(c) + 0, m_C(c)) \\ \iff & m_R(c) = m_C(c). \end{aligned}$$

- 2. Fall:
$$\begin{aligned} & \max(m_Q(c), m_P(c)) \leq m_R(c) \leq \min(m_Q(c) + m_P(c), m_C(c)) \\ \iff & \max(m_C(c), m_C(c)) \leq m_R(c) \leq \min(m_C(c) + m_C(c), m_C(c)) \\ \iff & m_R(c) = m_C(c). \end{aligned}$$

Für die Transformation der Differenzoperation `sExp -- sExp'` ergibt sich in der Kernsprache gemäß Definition 5.22:

- 1. Fall:
$$\begin{aligned} & \max(0, m_Q(c) - m_P(c)) \leq m_R(c) \leq \\ & \qquad\qquad m_Q(c) - \max(0, m_Q(c) + m_P(c) - m_C(c)) \\ \iff & \max(0, m_C(c) - 0) \leq m_R(c) \leq m_C(c) - \max(0, m_C(c) + 0 - m_C(c)) \\ \iff & m_R(c) = m_C(c). \end{aligned}$$
- 2. Fall:
$$\begin{aligned} & \max(0, m_Q(c) - m_P(c)) \leq m_R(c) \leq \\ & \qquad\qquad m_Q(c) - \max(0, m_Q(c) + m_P(c) - m_C(c)) \\ \iff & \max(0, m_C(c) - m_C(c)) \leq m_R(c) \leq \\ & \qquad\qquad m_C(c) - \max(0, m_C(c) + m_C(c) - m_C(c)) \\ \iff & m_R(c) = 0. \end{aligned}$$

Für die Transformation der Schnittmengenoperation `sExp ** sExp'` ergibt sich in der Kernsprache gemäß Definition 5.22:

- 1. Fall:
$$\begin{aligned} & \max(0, m_Q(c) + m_P(c) - m_C(c)) \leq m_R(c) \leq \min(m_Q(c), m_P(c)) \\ \iff & \max(0, m_C(c) + 0 - m_C(c)) \leq m_R(c) \leq \min(m_C(c), 0) \\ \iff & m_R(c) = 0. \end{aligned}$$
- 2. Fall:
$$\begin{aligned} & \max(0, m_Q(c) + m_P(c) - m_C(c)) \leq m_R(c) \leq \min(m_Q(c), m_P(c)) \\ \iff & \max(0, m_C(c) + m_C(c) - m_C(c)) \leq m_R(c) \leq \min(m_C(c), m_C(c)) \\ \iff & m_R(c) = m_C(c). \end{aligned}$$

PFADAUSDRÜCKE IN MENGENAUSDRÜCKEN INNERHALB VON CONSTRAINTS: Als Nächstes betrachten wir Pfadausdrücke, die in Mengenausdrücken innerhalb von Constraints vorkommen, und daher potentiell das Schlüsselwort **`dref`** enthalten. Eine Clafer-Spezifikation, die in der Kernsprache vorliegt, darf gemäß Definition 6.4 nur kontextunabhängige Pfadausdrücke `pExpr` $\in \langle pathExpr \rangle$ aufweisen, welche die Form `path` oder `path.dref` haben, wobei der Teilpfad `path` referenzfrei ist. Gemäß Definition 5.16 hat dies zur Konsequenz, dass ein Pfad p, der durch den Pfadausdruck `pExpr` repräsentiert wird und die Auswertungsschritte $W^\intercal = (W_1^\intercal, \ldots, W_m^\intercal)$ aufweist, nur für den Auswertungsschritt $W_m^\intercal$ aus mehr als einer Kante $W_M^\intercal = \{\, c_q \xrightarrow{r} c, c_q \xrightarrow{r} c', \ldots \}$ bestehen kann. Zudem gelten die Zusammenhänge $c_q \twoheadrightarrow$ `sExp` und $\{\, c, c', \ldots \} = \psi(c_q)$. Für die Auswertungsschritte $W_i^\intercal$ mit $1 \leq i \leq m-1$ gilt wie zuvor, dass diese jeweils aus genau einer Kante bestehen, welche die Schachtelungshierarchie im Abhängigkeitsgraphen traversieren. Mit Definition 5.17 ergeben sich durch die Transformation des Pfadausdrucks folgende Bedingungen mit $b_l, b_g \in \mathcal{L}_{\mathbb{B}}$:

$$
\begin{aligned}
T_M^{ms}(\texttt{path.}\textbf{dref})(W_i^\intercal) &:= M_P \quad \text{Supp}(M_P) = \{\, c, c', \dots \,\} \\
T_M^{ms}(\texttt{path})(W_{i-1}^\intercal) &:= M_Q \quad \text{Supp}(M_Q) = \{\, c_q \,\} \\
&\qquad\quad W_i^\intercal = \{\, c_q \xrightarrow{r} c, c_q \xrightarrow{r} c', \dots \,\} \text{ mit } c_s \blacklozenge\!\!\rightarrow c_q \\
&\qquad\quad b_l := m_C(c_s) \leq |M_P| \leq m_Q(c_q) \qquad (1) \\
&\qquad\quad b_g := M_P = M_{Ref} \text{ mit } M_{Ref} = \text{msetRef}(c_q). \qquad (2)
\end{aligned}
$$

Wir setzen gemäß Definition 6.4 für Clafer-Spezifikationen, die in der Kernsprache vorliegen, voraus, dass die Clafer-Definitionen $c_s \in \{\, c_s \mid c_s \blacklozenge\!\!\rightarrow^+ c_q \,\}$, die in der Schachtelungshierarchie zur referenzierenden Clafer-Definition c_q übergeordnet sind, höchstens einmal instanziierbar sind und somit die Bedingung $\lambda_{max}(c_s) \leq 1$ für $c_s \blacklozenge\!\!\rightarrow c_q$ gilt. Für die Ober- und Untergrenze der transformierten Multimenge $T_M^{ms}(\texttt{sExp}) := M_{Ref}$ ergibt sich in diesem Fall gemäß Definition 5.11 zusammen mit Gleichung (2) folgende Bedingung $b \in \mathcal{L}_{\mathbb{B}}$:

$$
\begin{aligned}
b &:= \left|M_{Ref}\right| \leq m_C(c_q) \leq m_C(c_s) \cdot \left|M_{Ref}\right| \\
&\iff \left|M_{Ref}\right| = m_C(c_q) \\
&\iff m_C(c) + m_C(c') + \ldots = m_C(c_q). \qquad (3)
\end{aligned}
$$

Durch die vorherige Betrachtung der Transformation von Mengenausdrücken, in denen nur referenzfreie Pfadausdrücke (ohne das **dref**-Schlüsselwort) auftreten, können wir schließen, dass die Multimenge M_{Ref} eine Zählfunktion mit globalen Zählfunktionswerten aufweist, sodass die Bedingung $\forall c \in \text{Supp}(M_{Ref}) : m_{Ref}(c) = m_C(c)$ gilt. In Gleichung (3) kann somit der Ausdruck $\left|M_{Ref}\right|$ durch die Summe der globalen Zählfunktionswerte ersetzt werden. Wegen Gleichung (2) kann geschlossen werden, dass eine Multimenge M_Q, die aus der Transformation von Pfadausdrücken `path.dref` resultiert, für alle Elemente der Trägermenge $\text{Supp}(M_Q)$ immer globale Zählfunktionswerte $m_C(c)$ aufweist.

AUSDRÜCKE MIT MENGENVERGLEICH: Einen Ausdruck, in dem zwei Mengenausdrücke $\texttt{sExp}, \texttt{sExp}' \in \langle setExpr \rangle$ verglichen werden, transformieren wir gemäß Definition 5.28, indem wir zunächst die Mengenausdrücke zu den Multimengen $T_M^{ms}(\texttt{sExp}) := M_P$ und $T_M^{ms}(\texttt{sExp}') := M_Q$ transformieren. Anschließend werden Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ eingeführt, die Zählfunktionen der Multimengen M_P und M_Q elementweise für alle Clafer-Definitionen $c \in \text{Supp}(M_P)$ in Beziehung setzen.

Gemäß der bisherigen Betrachtungen nehmen wir an, dass die Multimengen M_P und M_Q für alle Clafer-Definitionen der Trägermenge $c \in \text{Supp}(M_P)$ bzw. $c \in \text{Supp}(M_Q)$ globale Zählfunktionswerte $m_C(c)$ aufweisen. Für Ausdrücke mit Mengenvergleich können entsprechend folgende zwei Fälle auftreten:

1. Fall: Eine Clafer-Definition $c \in C$ ist in der Trägermenge $\text{Supp}(M_Q)$, jedoch nicht in der Trägermenge von $\text{Supp}(M_P)$ enthalten.

2. Fall: Eine Clafer-Definition $c \in C$ ist in der Trägermenge von Multimenge M_Q und in der Trägermenge von Multimenge M_P enthalten.

Für beide Fälle zeigen wir im Folgenden exemplarisch für die Transformation des Mengenvergleichsausdrucks $T^{ms}_{\mathbb{B}}(\texttt{sExp}\ \texttt{in}\ \texttt{sExp}')$, dass diese für Clafer-Spezifikationen, die in der Kernsprache vorliegen, zu Aussagen über globale Zählfunktionswerte der Multimenge M_C führen. Es ergibt sich gemäß Definition 5.28 Folgendes:

1. Fall:

$$\begin{aligned} & m_P(c) \leq m_Q(c) \\ \iff & m_C(c) \leq 0 \\ \iff & m_C(c) = 0. \end{aligned}$$

2. Fall:

$$\begin{aligned} & m_P(c) \leq m_Q(c) \\ \iff & m_C(c) \leq m_C(c) \\ \iff & m_C(c) = m_C(c). \end{aligned}$$

AUSDRÜCKE MIT EINFACHER QUANTIFIZIERUNG ÜBER MENGEN: Ausdrücke mit einfacher Quantifizierung über Mengen transformieren wir gemäß Definition 5.26 in Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ der Multimengen-Repräsentation, indem wir zunächst den quantifizierten Mengenausdruck $\texttt{sExp} \in \langle setExpr \rangle$ durch Anwendung der Funktion T^{ms}_M zu einer Multimenge $T^{ms}_M(\texttt{sExp}) := M_P$ transformieren und anschließend Bedingungen über die Zählfunktionswerte formulieren. Entsprechend der bisherigen Betrachtungen nehmen wir an, dass die transformierte Multimenge M_P nur globale Zählfunktionswerte aufweist, sodass die Bedingung $\forall c \in \text{Supp}(M_P) : m_P(c) = m_C(c)$ gilt. Der Betrag der Multimenge M_P ermittelt sich demnach wie folgt:

$$|M_P| = m_C(c) + m_C(c') + \ldots$$

Für die Transformation von Ausdrücken mit einfacher Quantifizierung **some**, **lone**, **one** und **no** ergibt sich somit gemäß Definition 5.26 in der Kernsprache Folgendes:

$$b := m_C(c) + m_C(c') + \ldots \left\{ \begin{matrix} \geq \\ \leq \\ = \end{matrix} \right\} 1. \qquad b := m_C(c) + m_C(c') + \ldots = 0.$$

Die Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ bestehen somit in der Kernsprache nur aus Zählfunktionswerten der globalen Multimenge M_C.

AUSSAGENLOGISCHE AUSDRÜCKE: Aussagenlogische Ausdrücke transformieren wir gemäß Definition 5.25 durch Anwendung der Funktion $T^{ms}_{\mathbb{B}}$ in Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ der Multimengen-Repräsentation. Hierdurch werden die transformierten Bedingungen, die entsprechend der bisherigen Betrachtungen nur globale Zählfunktionswerte enthalten, durch Negationen, Konjunktionen und Disjunktionen zueinander in Beziehung gesetzt. Die resultierenden verknüpften Bedingungen enthalten daher auch nach der Transformation nur Zählfunktionswerte der globalen Multimenge M_C.

NUMERISCHE AUSDRÜCKE: Numerische Ausdrücke $\texttt{nExp} \in \langle \mathit{numExpr} \rangle$ transformieren wir durch Anwendung der Funktion T_n^{ms} in arithmetische Formeln $l \in \mathcal{L}_n$ der Multimengen-Repräsentation. Ein Pfadausdruck, der innerhalb eines numerischen Ausdrucks vorkommt und auf eine Hilfs-Clafer-Definition $c_z \in C_{\mathbb{Z}}$ oder $c_z \in C_{\mathbb{R}}$ zeigt, wird gemäß Definition 5.17 folgendermaßen transformiert:

$$\begin{aligned} T_n^{ms}(\texttt{path.}\textbf{dref})(W_i^{\mathsf{T}}) &:= v && \text{mit } v = \text{numRef}(c_z) \text{ und } W_i^{\mathsf{T}} = \{\, c_q \xrightarrow{r} c_z \,\}, \\ T_M^{ms}(\texttt{path})(W_{i-1}^{\mathsf{T}}) &:= M_Q && \text{mit } W_{i-1}^{\mathsf{T}} = \{\, c_s \xrightarrow{n} c_q \,\}. \end{aligned}$$

Jedem Pfadausdruck wird demnach durch die Funktion $\text{numRef}(c_z)$ gemäß Definition 5.11 eine Variable $v \in \mathbb{V}$ zugeordnet, die von einer Clafer-Definition referenziert wird, die ein ganzzahliges oder reellwertiges Attribut repräsentiert. Es gilt somit der Zusammenhang $c_q \twoheadrightarrow c_z$. Da Clafer-Spezifikationen, die in der Kernsprache vorliegen, nur Pfadausdrücke enthalten dürfen, die kontextunabhängig sind, repräsentiert die Variable v alle zulässigen Werte, die eine Clafer-Definition c_q referenzieren darf. Zusätzliche Einschränkungen durch arithmetische Operationen, welche die Variable v einbeziehen, schränken dementsprechend immer alle möglichen Werte ein, die durch eine Clafer-Definition c_q referenzierbar sind. Die transformierten Variablen $v \in \mathbb{V}$ gelten somit über alle Clafer-Definitions-Instanzen hinweg innerhalb der transformierten numerischen Ausdrücke.

Zusammenfassend haben wir in diesem Beweisschritt gezeigt, dass alle Sprachbestandteile der Kernsprache in der Multimengen-Repräsentation immer zu Bedingungen $b \in \mathcal{L}_{\mathbb{B}}$ über ausschließlich globale symbolische Zählfunktionswerte $m_C(c)$ der Multimenge M_C übersetzt werden. Constraints, die in der Kernsprache formuliert sind, schränken somit stets alle Instanzen einer Clafer-Definition ein.

Beweisschritt 2: Konstruktion von Clafer-Spezifikationsinstanzen

In diesem Abschnitt beschreiben wir ein Verfahren zur Konstruktion einer Clafer-Spezifikationsinstanz $is \in [\![cs]\!]$ aus einer konkreten Multimengen-Belegung $\mathfrak{M}_C$ einer (symbolischen) Multimengen-Repräsentation $ms \in MS$. Gemäß Definition 4.20 wird eine Clafer-Spezifikationsinstanz $is \in [\![cs]\!]$ durch eine Menge von Clafer-Definitions-Instanzen $is = \{i_1, i_2, \ldots\}$ repräsentiert, wobei jede Clafer-Definitions-Instanz genau einer Clafer-Definition $c \in C$ zugeordnet ist. Wir schreiben $is(c)$, um uns auf die Teilmenge der Clafer-Definitions-Instanzen einer Clafer-Spezifikationsinstanz zu beziehen, die einer Clafer-Definition c zugeordnet ist.

Wie in Abschnitt 4.3.1 beschrieben, besteht eine konkrete Clafer-Spezifikationsinstanz aus einer Menge von Instanzen von Clafer-Definitionen, wobei jede Instanz (außer der Instanz, die zur Wurzel-Clafer-Definition c_r korrespondiert) genau einer Instanz der übergeordnete Eltern-Clafer-Definition zugeordnet ist. Zur Konstruktion einer Clafer-Spezifikationsinstanz erzeugen wir ausgehend von der Wurzel-Clafer-Definition c_r anhand der konkreten Multimengen-Belegung $\mathfrak{M}_C$ Instanzen von Clafer-Definitionen

durch wiederholtes Anwenden der nachfolgenden Konstruktionsschritte. Wir schreiben $\mathfrak{M}_C(c) = k$, um uns auf die Anzahl von k Instanzen einer Clafer-Definition $c \in C$ in einer Multimengen-Belegung $\mathfrak{M}_C$ zu beziehen.

Schritt 1: Ausgehend von der Wurzel-Clafer-Definition c_r wird eine Clafer-Definitions-Instanz erzeugt und der Clafer-Definition c_r zugeordnet. Per Definition 5.8 ist gefordert, dass die Bedingung $\mathfrak{M}_C(c_r) = 1$ immer erfüllt ist. Anschließend wird mit Schritt 2 fortgefahren, wobei hierbei Clafer-Definition c_r betrachtet wird.

Schritt 2: Für alle Kind-Clafer-Definitionen $c' \in C$ der gerade betrachteten Eltern-Clafer-Definition $c \in C$ mit $c \blacklozenge\!\!\rightarrow c'$ wird die Anzahl $\mathfrak{M}_C(c') = k$ an Clafer-Definitions-Instanzen erzeugt. Anschließend werden die erzeugten Instanzen gleichmäßig auf alle Instanzen der Eltern-Clafer-Definition c verteilt, sodass jeder Instanz der Eltern-Clafer-Definition c mindestens $\lfloor \mathfrak{M}_C(c')/\mathfrak{M}_C(c) \rfloor$ Instanzen der Clafer-Definition c' zugeordnet sind.

Falls eine nicht durch $\mathfrak{M}_C(c)$ ohne Rest teilbare Anzahl von Clafer-Definitions-Instanzen verteilt werden muss, wird der ungerade Anteil ($\mathfrak{M}_C(c')$ mod $\mathfrak{M}_C(c)$) ebenfalls gleichmäßig den Instanzen der Eltern-Clafer-Definition c zugeordnet. Nach Ausführung dieses Konstruktionsschrittes gilt $is(c') = \mathfrak{M}_C(c') = k$. Anschließend wird Schritt 2 für alle Kind-Clafer-Definitionen $c'' \in \{\, c'' \mid c' \blacklozenge\!\!\rightarrow c'' \,\}$ der zuvor betrachteten Kind-Clafer-Definition c' wiederholt.

Das beschriebene Verfahren ermöglicht es, aus einer konkreten Multimengen-Belegung eine Clafer-Spezifikationsinstanz zu konstruieren. Wie im vorherigen Abschnitt nachgewiesen, gelten für Spezifikationen, die in der Kernsprache vorliegen, Einschränkungen durch Constraints immer für die Menge aller Instanzen einer Clafer-Definition. Zudem führen Einschränkungen durch Multiplizitätsintervalle von Clafer-Definitionen dazu, dass die Anzahl der Instanzen einer Kind-Clafer-Definition c' multiplikativ von der Anzahl der Eltern-Clafer-Definition c mit $c \blacklozenge\!\!\rightarrow c'$ abhängt (dieser Zusammenhang wurde in Definition 5.8 als Bedingungen zwischen den globalen symbolischen Zählfunktionswerten $m_C(c)$ und $m_C(c')$ codiert). Es existiert somit immer mindestens eine zulässige Instanz einer Clafer-Spezifikation (die in der Kernsprache vorliegt) in der Instanzen einer Clafer-Definition gleichmäßig nach dem oben gezeigten Verfahren auf Instanzen ihrer Eltern-Clafer-Definition verteilt sind. Falls dies nicht immer der Fall wäre, müssten Constraints in der Kernsprache formulierbar sein, die nur Teil-Instanzmengen einer Clafer-Definition einschränken.

Zusammenfassend haben wir in diesem Beweisschritt ein Konstruktionsverfahren gezeigt, das zulässige Clafer-Spezifikationsinstanzen anhand einer konkreten Multimengen-Belegung erzeugt.

Beweisschritt 3: Nachweis der Vollständigkeit für Kernsprache

Mit der in Beweisschritt 1 gezeigten Übersetzung von Clafer-Spezifikationen, die in der Kernsprache vorliegen, in eine Multimengen-Repräsentation und dem in Beweisschritt 2 präsentierten Konstruktionsverfahren zur Übersetzung konkreter Multimengen-Belegungen in eine Clafer-Spezifikationsinstanz lässt sich nun Satz 6.5 beweisen: Ist eine Clafer-Spezifikation in der Kernsprache inkonsistent, so besitzt die zugehörige Multimengen-Repräsentation keine Lösung.

Durch Widerspruch lässt sich nun beweisen: Würde die zugehörige Multimengen-Repräsentation eine Lösung besitzen, so könnte man zu dieser Lösung nach dem obigen Konstruktionsverfahren eine Instanz der Clafer-Spezifikation erzeugen. Damit wäre die Clafer-Spezifikation jedoch nicht inkonsistent. Jede Inkonsistenz einer Clafer-Spezifikation, die in der Kernsprache vorliegt, kann somit durch unser Analyseverfahren identifiziert werden. □

6.1.4 *Semantische Eigenschaften*

Eine Clafer-Spezifikation $cs \in CS$, die in der erweiterten Kernsprache vorliegt, kann in eine Multimengen-Repräsentation $ms := T^{ms}(cs)$ transformiert werden. Eine Multimengen-Belegung $m \in [\![ms]\!]$ ist gültig, falls alle strukturellen Eigenschaften und alle zusätzlich eingeführten Bedingungen zwischen Zählfunktionen der Multimengen erfüllt sind. Die Multimengen-Repräsentation ermöglicht die korrekte Analyse einer Clafer-Spezifikation $cs \in CS^{k+}$ und die vollständige Analyse einer Clafer-Spezifikation $cs \in CS^{k}$ auf semantische Eigenschaften, wie sie in Abschnitt 3.2.2 vorgestellt wurden. Die Definition der semantischen Eigenschaften auf Basis der Multimengen-Repräsentation $ms = (M_C, \mathbb{M}_C)$ und der Bedingungen zwischen Zählfunktionen, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ formuliert sind, lautet wie folgt:

Definition 6.6 (Semantische Eigenschaften von Clafer-Spezifikationen)

- *Unbeschränktheit einer Clafer-Definition*: Eine Clafer-Definition $c \in C$ ist *semantisch unbeschränkt* für einen beliebigen Wert $n \in \mathbb{N}_0$ genau dann, wenn eine gültige Multimengen-Belegung für M_C mit $m_C(c) > n$ existiert.

- Anomalie vom Typ *fälschlicherweise unbeschränkt*: Eine Clafer-Definition $c \in C$ ist *fälschlicherweise unbeschränkt*, wenn c syntaktisch, jedoch nicht semantisch unbeschränkt ist.

- Anomalie vom Typ *totes Kardinalitätsintervall*: Eine Clafer-Definition $c \in C$ weist eine Anomalie vom Typ *totes Kardinalitätsintervall* auf dem Intervall $I := (l, u)$ auf (gegeben entweder durch ein Multiplizitätsintervall $\lambda_c(c)$ oder durch ein Gruppenkardinalitätsintervall $\lambda_g(c)$), falls es keine Multimengen-Belegung

gibt, für die ein Wert $k \in \mathbb{N}$ existiert, sodass die Bedingungen $l \leq k \leq u$ und $m_C(c) = k$ gelten.

- Anomalie vom Typ *tote Clafer-Definition*: Eine Clafer-Definition $c \in C$ weist eine Anomalie vom Typ *tote Clafer-Definition* auf, wenn das gesamte Multiplizitätsintervall $\lambda_c(c)$ eine Anomalie vom Typ *totes Kardinalitätsintervall* aufweist.

- Anomalie vom Typ *Kern-Clafer-Definition*: Eine Clafer-Definition $c \in C$ besitzt eine Anomalie vom Typ Kern-Clafer-Definition, falls sie syntaktisch optional definiert ist, aber jede konsistente Instanz der betrachteten Clafer-Spezifikation mindestens eine Instanz von c enthält. Die Clafer-Definition weist entsprechend auf dem Multiplizitätsintervall $(0, 0)$ eine Anomalie vom Typ *totes Kardinalitätsintervall* auf. Jede syntaktisch optionale Clafer-Definition, die eine Anomalie vom Typ *tote Clafer-Definition* aufweist, besitzt damit gleichzeitig auch eine Anomalie vom Typ *Kern-Clafer-Definition*.

Im Gegensatz zu existierenden Ansätzen zur Konsistenzprüfung von Clafer-Spezifikationen können durch unser Analyseverfahren Clafer-Spezifikationen (die in der erweiterten Kernsprache vorliegen) auf semantische Unbeschränktheit überprüft werden, da für die Transformation T^{ms} einer Clafer-Spezifikation keine expliziten Schranken zur Begrenzung des Suchraums angegeben werden müssen. Falls tatsächlich für eine Clafer-Spezifikation, die in der Kernsprache vorliegt, keine gültige Clafer-Spezifikationsinstanz besteht, gibt es somit ebenso keine konkrete Multimenge, welche die Bedingungen der Multimengen-Repräsentation erfüllt. Unser Analyseverfahren ist daher vollständig für alle Clafer-Spezifikationen, die in der Kernsprache spezifiziert sind.

Bei der Übersetzung einer beliebigen Clafer-Spezifikation $cs \in CS$ in die erweiterte Kernsprache CS^{k+} müssen gegebenenfalls Constraints entfernt werden, die Sprachelemente beinhalten, die in Abschnitt 6.1.1 gekennzeichnet wurden und durch unser Analyseverfahren nicht analysierbar sind. Durch das Entfernen von Constraints aus der Clafer-Spezifikation cs wird die Menge gültiger Clafer-Spezifikationsinstanzen potentiell vergrößert. Unser Ziel ist es nun, die Gültigkeit des folgenden Satzes zu zeigen:

Satz 6.7 (Korrektheit des Analyseverfahrens) Besitzt eine Multimengen-Repräsentation einer auf die erweiterte Kernsprache reduzierte Clafer-Spezifikation keine Lösung bzw. die Anomalie α, so besitzt die ursprüngliche nicht-reduzierte Clafer-Spezifikation ebenfalls keine Lösung bzw. eine Anomalie α.

Beweisskizze

Sei cs eine Clafer-Spezifikation und α eine Menge zusätzlicher Constraints, dann ist $[\![cs \oplus \alpha]\!]$ die um die Constraint-Menge α erweiterte Clafer-Spezifikation. Weiterhin gilt folgender Zusammenhang:

$$[\![cs \oplus \alpha]\!] \subseteq [\![cs]\!]. \tag{6.1}$$

Die zusätzliche Constraint-Menge α führt dazu, dass sich die spezifizierte Instanzmenge reduziert oder, falls die Constraint-Menge α zu den übrigen Constraint der Spezifikation redundant ist, gleich bleibt.

Außerdem sei $cs := cs_k \oplus \beta$ eine Clafer-Spezifikation, die aus einer Clafer-Spezifikation cs_k der erweiterten Kernsprache CS^{k+} und einer Menge von Constraints β besteht. Schließlich sei α eine Constraint-Menge, die eine bestimmte Anomalie charakterisiert. Dann gilt:

$$\begin{aligned} cs_k \text{ hat Anomalie } \alpha : &\iff [\![cs_k \oplus \alpha]\!] = \emptyset \\ &\overset{6.1}{\Longrightarrow} [\![cs_k \oplus \beta \oplus \alpha]\!] = [\![cs \oplus \alpha]\!] = \emptyset : \iff cs \text{ hat Anomalie } \alpha \quad \square \end{aligned}$$

Falls somit für die modifizierte, in der erweiterten Kernsprache befindliche Clafer-Spezifikation $cs_k \oplus \alpha$ geschlossen werden kann, dass keine gültige Multimengen-Belegung existiert, dann gibt es auch für die Clafer-Spezifikation $cs \oplus \alpha$ (die potentiell nicht analysierbare Sprachelemente enthält) keine Clafer-Spezifikationsinstanz. In diesem Sinne ist unsere Analyseverfahren korrekt für jede beliebige Clafer-Spezifikation, da jede identifizierte Anomalie oder Inkonsistenz einer in die erweiterte Kernsprache umgewandelten Clafer-Spezifikation $cs_k \in CS^{k+}$ zu einer tatsächlichen Anomalie der ursprünglichen Clafer-Spezifikation $cs \in CS$ korrespondiert.

6.2 ANALYSE MITTELS MATHEMATISCHER PROGRAMMIERUNG

In diesem Abschnitt beschreiben wir, wie wir Techniken der mathematischen Programmierung einsetzen, um gültige minimale bzw. maximale Belegungen der in Kapitel 5 vorgestellten Multimengen-Repräsentation einer Clafer-Spezifikation zu finden. Wie in Abschnitt 4.3 beschrieben, besteht ein (reguläres) mathematisches Optimierungsproblem aus Entscheidungsvariablen, einer Zielfunktion und Constraints in Form linearer Ungleichungen. Die Ungleichungen beschreiben hierbei einen konvexen Suchraum, der durch die Entscheidungsvariablen aufgespannt wird. Die Zielfunktion codiert die zu analysierenden semantischen Eigenschaften, die in Abschnitt 6.1.4 beschrieben wurden. Im Folgenden beschreiben wir, wie zunächst die Multimengen-Repräsentation in ein erweitertes Optimierungsproblem $ip' \in IP'$ transformiert wird. Das Vorgehen lehnt sich an dem in [121] skizzierten Modellierungsansatz für diskrete Optimierungsprobleme an. Das erweiterte Optimierungsproblem dient als Zwischenrepräsentation und

beinhaltet im Gegensatz zu regulären mathematischen Optimierungsproblemen Sprachelemente zur aussagenlogischen Verknüpfung von Constraints. In einem nächsten Schritt beschreiben wir, wie das erweiterte Optimierungsprobleme $ip' \in IP'$ in ein reguläres Optimierungsproblem $ip \in IP$ übersetzt wird, das durch etablierte Lösungsverfahren analysiert werden kann. Schließlich zeigen wir, wie die in Abschnitt 6.1.4 beschriebenen semantischen Eigenschaften als Zielfunktion des Optimierungsproblems repräsentiert werden.

6.2.1 *Formulierung als erweitertes Optimierungsproblem*

Wir formulieren die Suche nach einer minimal bzw. maximal gültigen Belegung der Multimengen-Repräsentation als gemischt-ganzzahliges lineares Optimierungsproblem (engl. Mixed Integer Linear Program - MILP [154]). Unsere in Abschnitt 4.3.2 eingeführte Multimengen-Repräsentation besteht aus der Multimenge M_C, welche die Anzahl von instanziierten Clafer-Definitionen einer Spezifikation angibt, weiteren Multimengen $M_P \in \mathbb{M}_C$ sowie Bedingungen zwischen (symbolischen) Zählfunktionswerten der Multimengen, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ formuliert sind. Das erweiterte Optimierungsproblem ermöglicht es, aussagenlogische Verknüpfungen von Constraints oder Entscheidungsvariablen zu formulieren. Die aussagenlogischen Sprachelemente von $\mathcal{L}_{\mathbb{B}}$ können somit ohne weitere Umformungen in ein erweitertes Optimierungsproblem übersetzt werden. Wir führen folgende Mengen ein, welche die Entscheidungsvariablen des Optimierungsproblems enthalten:

- Eine Entscheidungsvariable $x_c^C \in \mathbb{N}_0$ der Menge X^C gibt den Wert der Zählfunktion $m_C(c)$ für die Clafer-Definition c der Multimenge M_C an. Falls $x_c^C = k$, ist die Anzahl von Instanzen der Clafer-Definition c in der Multimenge M_C gleich k. Für Hilfs-Clafer-Definitionen $c_z \in C_{\mathbb{Z}} \cup C_{\mathbb{R}}$, die in Referenzen $c \twoheadrightarrow c_z$ positive oder negative ganzzahlige oder reelle Werte repräsentieren, führen wir separate Entscheidungsvariablen mit passenden Wertebereichen ein. Die Menge X^C ist definiert als:
$$X^C := \{\, x_c^C \mid x_c^C \in \mathbb{N}_0 \wedge c \in C \setminus (C_{\mathbb{Z}} \cup C_{\mathbb{R}}) \,\}.$$

- Analog gibt die Entscheidungsvariable x_c^P der Menge X^P den Wert der Zählfunktion $m_P(c)$ für die Clafer-Definition $c \in \mathrm{Supp}(M_P)$ der Multimenge M_P an. Die Menge der Entscheidungsvariablen X^P ist definiert als:
$$X^P := \{\, x_c^P \mid x_c^P \in \mathbb{N}_0 \wedge c \in \mathrm{Supp}(M_P) \,\}.$$
Für jede Multimenge $M_p \in \mathbb{M}_C$ wird eine entsprechende Menge X^P von Entscheidungsvariablen eingeführt.

- Eine Variable $v \in V_{\mathbb{Z}}$ der Sprache $\mathcal{L}_n$ repräsentiert in der Multimengen-Repräsentation einen ganzzahligen positiven oder negativen Wert einer Hilfs-Clafer-Definition

$c_z \in C_{\mathbb{Z}}$, die in Referenzen $c \twoheadrightarrow c_z$ auftritt. Eine Entscheidungsvariable $x_v^{V_{\mathbb{Z}}}$ der Menge $X^{V_{\mathbb{Z}}}$ gibt den ganzzahligen Wert einer Variablen $v \in V_{\mathbb{Z}}$ an. Die Menge $X^{V_{\mathbb{Z}}}$ ist definiert als:

$$X^{V_{\mathbb{Z}}} := \{ x_v^{V_{\mathbb{Z}}} \mid x_v^{V_{\mathbb{Z}}} \in \mathbb{Z} \wedge v \in V_{\mathbb{Z}} \}.$$

- Analog repräsentiert eine Variable $v \in V_{\mathbb{R}}$ der Sprache $\mathcal{L}_n$ in der Multimengen-Repräsentation einen reellen positiven oder negativen Wert einer Hilfs-Clafer-Definition $c_z \in C_{\mathbb{R}}$. So gibt eine Entscheidungsvariable $x_v^{V_{\mathbb{R}}}$ der Menge $X^{V_{\mathbb{R}}}$ den positiven oder negativen reellen Wert einer Variablen $v \in V_{\mathbb{R}}$ der Sprache $\mathcal{L}_n$ an. Die Menge der Entscheidungsvariablen $X^{V_{\mathbb{R}}}$ ist definiert als:

$$X^{V_{\mathbb{R}}} := \{ x_i^{V_{\mathbb{R}}} \mid x_i^{V_{\mathbb{R}}} \in \mathbb{R} \wedge i \in V_{\mathbb{R}} \}.$$

- Zusätzlich führen wir reellwertige Hilfsvariablen $h \in H_{\mathbb{R}}$, ganzzahlige Hilfsvariablen $h \in H_{\mathbb{Z}}$ und binäre Hilfsvariablen $h \in H_{\mathbb{B}}$ zur Übersetzung des erweiterten in das reguläre mathematische Optimierungsproblem ein.

Arithmetische Operationen mit Zählfunktionen von Multimengen und Variablen, die in der Sprache $\mathcal{L}_n$ der Multimengen-Repräsentation angegeben sind, formulieren wir als Terme innerhalb von Constraints im erweiterten mathematischen Optimierungsproblem wie folgt:

- Die Addition $A + B$ mit $A, B \in \mathcal{L}_n$ von arithmetischen Ausdrücken entspricht der Addition von Termen im Optimierungsproblem.

- Die Multiplikation einer Konstanten mit einer Variablen entspricht im Optimierungsproblem der Multiplikation eines Parametern mit einer Entscheidungsvariablen. Die Linearität der Terme ist auf Sprachebene sichergestellt, da die Multiplikation von Variablen, die zu einer nicht-linearen Kombination führen würde, in der Sprache $\mathcal{L}_n$ nicht zulässig ist.

- Zur Codierung des Minimums $\min(A, B)$ zweier arithmetischer Ausdrücke der Sprache $\mathcal{L}_n$ ersetzen wir im Optimierungsproblem den Ausdruck durch eine Hilfsvariable $h \in H_{\mathbb{R}}$ und führen folgenden Constraint ein:

  $$(A \leq B \implies h = A) \wedge (A > B \implies h = B).$$

  Der Constraint enthält aussagenlogische Operatoren zur Verknüpfung von Teil-Constraints und muss in einem späteren Transformationsschritt daher in eine Menge von linearen Ungleichungen transformiert werden.

- Entsprechend ersetzen wir das Maximum $\max(A, B)$ zweier arithmetischer Ausdrücke der Sprache $\mathcal{L}_n$ im Optimierungsproblem durch eine Hilfsvariable $h \in H$ und führen folgenden Constraint ein:

  $$(A \geq B \implies h = A) \wedge (A < B \implies h = B).$$

Bedingungen zwischen (symbolischen) Zählfunktionswerten, die in der Sprache $\mathcal{L}_{\mathbb{B}}$ angegeben sind, formulieren wir als Constraints im erweiterten mathematischen Optimierungsproblem wie folgt:

- Die aussagenlogischen Formeln $A \leq B$, $A \geq B$ formulieren wir ohne weitere Umformungen als Constraints im erweiterten Optimierungsproblem, wobei A und B arithmetische Ausdrücke der Sprache $\mathcal{L}_n$ sind.
- Die aussagenlogischen Formeln $A < B$, $A > B$ und $A = B$ formen wir in die aussagenlogischen Formeln $\neg(A \geq B)$, $\neg(A \leq B)$ bzw. $A \geq B \wedge A \leq B$ um, wobei A und B arithmetische Ausdrücke der Sprache $\mathcal{L}_n$ sind.
- Die aussagenlogische Formel $A \implies B$ formen wir in die äquivalente aussagenlogische Formel $\neg A \vee B$ um.
- Aussagenlogische Formeln $\neg A$, $A \wedge B$, $A \vee B$ mit den aussagenlogischen Formeln A und B formulieren wir als Constraints im erweiterten Optimierungsproblem.

Folgendes Beispiel zeigt die Formulierung von Bedingungen zur Abschätzung der Schachtelungshierarchie der Multimengen-Repräsentation im erweiterten Optimierungsproblem.

Beispiel 6.8 (Formulierung der Bedingungen zur Abschätzung der Schachtelungsrelationen) In Abschnitt 5.2 wird die Schachtelungshierarchie $c' \blacklozenge\!\!\rightarrow c$ zwischen einer Clafer-Definition $c \in C$ mit Multiplizitätsintervall $(l, u) = \lambda_c(c)$ und ihrer übergeordneten Clafer-Definition $c' \in C$ als Bedingungen zwischen den Zählfunktionswerten $m_C(c)$ und $m_C(c')$ repräsentiert. Die Entscheidungsvariable x_c^C repräsentiert den Zählfunktionswert $m_C(c)$. Entsprechend repräsentiert die Entscheidungsvariable $x_{c'}^C$ den Zählfunktionswert $m_C(c')$. Falls das Multiplizitätsintervall unbeschränkt ist ($u = *$), führen wir im erweiterten gemischt-ganzzahligen mathematischen Optimierungsproblem folgenden Constraint ein:

$$l \cdot x_{c'}^C \leq x_c^C.$$

Falls das Multiplizitätsintervall nach oben hin beschränkt ist ($u \neq *$), führen wir zusätzlich folgenden Constraint ein, um die Entscheidungsvariable x_c^C einzuschränken:

$$x_c^C \leq u \cdot x_{c'}^C.$$

Entsprechend transformieren wir Bedingungen der Multimengen-Repräsentation zum Abbilden von Gruppenkardinalitätsintervallen $(l, u) = \lambda_g(c)$ einer Clafer-Definition $c \in C$ mit untergeordneten Clafer-Definitionen $c' \in \{c' \in C \mid c \blacklozenge\!\!\rightarrow c'\}$ in

Constraints des erweiterten gemischt-ganzzahligen mathematischen Optimierungsproblems. Die Summe der Entscheidungsvariablen $x^C_{c'}$, welche die Zählfunktionen $m_C(c')$ der untergeordneten Clafer-Definitionen c' repräsentieren, ist nach unten hin durch das Gruppenkardinalitätsintervall l eingeschränkt. Wir führen somit folgenden Constraint ein:

$$l \cdot x^C_c \leq \sum_{c' \in \{c' \in C | c \blacklozenge\!\!\rightarrow c'\}} x^C_{c'}.$$

Die Summe der Entscheidungsvariablen, die untergeordnete Clafer-Definitionen repräsentieren, ist nach oben hin eingeschränkt, falls die Bedingung $u \neq *$ gilt. In diesem Fall führen wir daher zusätzlich folgenden Constraint ein:

$$\sum_{c' \in \{c' \in C | c \blacklozenge\!\!\rightarrow c'\}} x^C_{c'} \leq u \cdot x^C_c.$$

Folgendes Beispiel demonstriert die Formulierung von Bedingungen in der Multimengen-Repräsentation zur Abschätzung der Untergrenze und Obergrenze einer Mengenvereinigung.

Beispiel 6.9 (Formulierung der Bedingungen zur Abschätzung von Mengenausdrücken) Zur Repräsentation der Vereinigung von zwei Mengenausdrücken wird eine Multimenge M_R als Ergebnis der Mengenoperation eingeführt. Die Multimengen M_P und M_Q beziehen sich jeweils auf die beiden Mengenausdrücke, die vereinigt werden. Zur Formulierung im erweiterten mathematischen Optimierungsproblem führen wir für jede Multimenge die Mengen X_R, X_P und X_Q ein, die für alle Clafer-Definition $c \in C$ Entscheidungsvariablen enthalten, welche die (symbolischen) Zählfunktionswerte der Multimengen repräsentieren. Für die in Kapitel 5 formulierte Bedingung $m_R(c) \leq \min(m_P(c), m_Q(c))$ (für alle $c \in C$) zur Abschätzung der Untergrenzen der Zählfunktion $m_R(c)$ führen wir im mathematischen Optimierungsproblem folgenden Constraint ein:

$$\underbrace{\left(x^P_c \leq x^Q_c \implies x^R_c \leq x^P_c\right)}_{\text{lhs}} \wedge \underbrace{\left(\neg \left(x^P_c \leq x^Q_c\right) \implies x^R_c \leq x^Q_c\right)}_{\text{rhs}}.$$

Entsprechend wurde in Abschnitt 5.5.1 zur Abschätzung der Obergrenze von $m_R(c)$ (falls nicht unbeschränkt) die Bedingung $\max(m_Q(c), m_P(c)) \leq m_R(c)$ verwendet. Zur Repräsentation dieser Bedingung im mathematischen Optimierungsproblem führen wir folgenden Constraint ein:

$$\underbrace{\left(x_c^P \geq x_c^Q \implies x_c^P \leq x_c^R\right)}_{\text{lhs}} \wedge \underbrace{\left(\neg\left(x_c^P \geq x_c^Q\right) \implies x_c^Q \leq x_c^R\right)}_{\text{rhs}}.$$

Die linke Bedingung (lhs) repräsentiert den Fall, dass die Zählfunktion $m_P(c)$ größer oder gleich $m_Q(c)$ ist, die Bedingung auf der rechten Seite (rhs) repräsentiert den entgegengesetzten Fall. Beide Bedingungen zusammen repräsentieren somit einen max-Operator.

6.2.2 *Formulierung als reguläres Optimierungsproblem*

Im vorherigen Abschnitt wurde beschrieben, wie die Multimengen-Repräsentation einer Clafer-Spezifikation als erweitertes Optimierungsproblem $ip' \in IP'$ formuliert wird. Das erweiterte Optimierungsproblem dient als Zwischenrepräsentation und erlaubt die aussagenlogische Verknüpfung von Constraints, wobei ein Constraint eine lineare Ungleichung ist. Abhängigkeiten zwischen aussagenlogischen Formeln der Sprache $\mathcal{L}_{\mathbb{B}}$ können so ohne Umformung der ursprünglichen Struktur in das erweiterte Optimierungsproblem überführt werden. Die Zielrepräsentation $ip \in IP$, die von etablierten MILP-Solvern auswertbar ist, erlaubt jedoch nur Constraints in Form linearer Ungleichungen, die implizit konjunktiv miteinander verknüpft sind (da sie alle gleichzeitig erfüllt sein müssen). Im Folgenden beschreiben wir, wie ein erweitertes Optimierungsproblem $ip' \in IP'$ in ein reguläres Optimierungsproblem $ip \in IP$ übersetzt wird.

Zur Übersetzung von Constraints, die aus aussagenlogisch verknüpften Teil-Constraints in Form lineare Ungleichungen bestehen, gehen wir folgendermaßen vor.

Schritt 1: Zunächst ersetzen wir die Teil-Constraints eines Constraints, die aussagenlogisch verknüpft sind, durch binäre Hilfsvariablen.

Schritt 2: In einem weiteren Schritt transformieren wir den Constraint, in dem Teil-Constraints durch binäre Hilfsvariablen ersetzt wurden, in die konjunktive Normalform (KNF).

Schritt 3: Im nächsten Schritt übersetzen wir jede Klausel der KNF in eine lineare Ungleichung, die dem regulären Optimierungsproblem als neue Constraint hinzugefügt wird.

Schritt 4: Zuletzt verknüpfen wir die jeweiligen Teil-Constraints mit der binären Hilfsvariablen, durch die sie in dem transformierten Constraint ersetzt wurden.

Die genannten Schritte werden im Weiteren genauer beschrieben.

Schritt 1: Ersetzen von Teil-Constraints durch binäre Hilfsvariablen

Einen Constraint, der aus mehreren aussagenlogisch verknüpften Teil-Constraints besteht, formen wir in diesem Schritt um, indem wir jeden Teil-Constraint durch eine neu eingeführte binäre Hilfsvariable $h \in H_{\mathbb{B}}$ ersetzen.

Schritt 2: Transformation in KNF

Der im vorherigen Schritt umgeformte Constraint wird in diesem Schritt in die konjunktive Normalform (KNF) übersetzt [143]. Eine aussagenlogische Formel befindet sich in konjunktiver Normalform, falls sie aus einer Konjunktion von Klauseln besteht. Die Klausel einer KNF ist hier die Disjunktion von Hilfsvariablen h oder negierten Hilfsvariablen $\neg h$ mit $h \in H_{\mathbb{B}}$.

Schritt 3: Übersetzung von Klauseln der KNF in Constraint

Jede Klausel $h \vee \ldots \vee \neg h' \vee \ldots$ der KNF wird im nächsten Schritt in einen Constraint übersetzt, der folgende Form hat:

$$h + \ldots + (1 - h') \geq 1.$$

Negierte Hilfsvariablen $\neg h'$ werden hierbei durch den äquivalenten Term $1 - h'$ ersetzt.

Schritt 4: Verknüpfung von Teil-Constraints mit Hilfsvariablen

In einem letzten Schritt werden die Hilfsvariablen des transformierten Constraints mit den Teil-Constraints verknüpft. Ein Teil-Constraint lässt sich auf eine lineare Ungleichung folgender Form zurückführen:

$$\underbrace{p x_c^R + \ldots + p' x_{c'}^R}_{lhs} \left\{ \begin{array}{c} \leq \\ \geq \end{array} \right\} \underbrace{p''}_{rhs}.$$

Die linke Seite der Ungleichung besteht aus einer Summe von Entscheidungsvariablen $x_c^R \in X^C \cup X^P \cup V_{\mathbb{Z}} \cup V_{\mathbb{R}}$, die mit Parametern $p \in \mathbb{R}$ gewichtet sind. Die rechte Seite besteht aus einer Konstanten p''. Im Folgenden bezeichnen wir die linke Seite der Ungleichung mit *lhs* und die rechte Seite der Ungleichung mit *rhs*. Wir nehmen an, dass jeder Teil-Constraint zuvor durch eine neue Hilfsvariable $h \in H_{\mathbb{B}}$ ersetzt wurde, mit welcher der Teil-Constraint nun verknüpft werden soll. Für den Zusammenhang zwischen dem Teil-Constraint und der Hilfsvariablen muss Folgendes gelten:

- Falls der Teil-Constraint erfüllt ist, muss die binäre Hilfsvariable den Wahrheitswert *wahr* repräsentieren. Dies entspricht der Bedingung $h = 1$.
- Falls der Teil-Constraint nicht erfüllt ist, muss die binäre Hilfsvariable den Wahrheitswert *falsch* repräsentieren. Dies entspricht der Bedingung $h = 0$.

Es ergibt sich somit folgende aussagenlogische Formel, um den Zusammenhang zwischen einem Teil-Constraint und der Hilfsvariablen zu charakterisieren:

$$h = 1 \Leftrightarrow lhs \leq rhs.$$

Die aussagenlogische Formel lässt sich zerlegen in folgende beide Formeln:

$$h = 1 \implies lhs \leq rhs \quad \text{und} \quad h = 0 \implies \neg(lhs \leq rhs).$$

Für jede Formel sind wir nun in der Lage, separate Constraints im regulären Optimierungsproblem zu formulieren.

Zur Repräsentation der Negation führen wir eine zusätzliche binäre Entscheidungsvariable $h' \in H_{\mathbb{B}}$ ein und stellen durch den zusätzlichen Constraint $h + h' = 1$ sicher, dass immer genau eine Hilfsvariable den Wahrheitswert wahr repräsentiert (entweder $h = 1 \wedge h' = 0$ oder $h = 0 \wedge h' = 1$).

Zur Transformation der ersten Formel $h = 1 \implies lhs \leq rhs$ verwenden wir Special Ordered Sets vom Typ 1 (SOS1) [18]. SOS1 können binäre, reellwertige oder ganzzahlige Entscheidungsvariablen enthalten. Für die Elemente eines SOS1 gilt, dass höchstens ein Element größer als null sein darf. Durch die Verwendung von SOS1 ist es somit möglich, nicht-lineare Abhängigkeiten zwischen binären und ganzzahligen bzw. reellwertigen Entscheidungsvariablen zu formulieren. Wir schreiben $\text{sos}(h, s)$ für einen SOS1 mit den Elementen h und s. Zusätzlich führen wir die reellwertigen Hilfsvariablen $s, s' \in H^{\mathbb{R}}$ ein.

Die erste Formel $h = 1 \implies lhs \leq rhs$ kann nun mithilfe von SOS1 im regulären Optimierungsproblem durch folgende Constraints ersetzt werden:

$$lhs \leq rhs + s \text{ mit } \text{sos}(h, s) \quad \text{und} \quad h + s \geq \epsilon.$$

Die Konstante $\epsilon \in \mathbb{R}$ ist hierbei ein beliebig kleiner Wert. Für die Erklärung der Funktionsweise der Constraints unterscheiden wir zwei Fälle:

Fall 1: Falls $h = 1$ gilt, muss für die Hilfsvariable s die Bedingung $s = 0$ gelten, da höchstens ein Element des SOS1 $\text{sos}(h, s)$ größer als null sein darf. Die Ungleichung $lhs \leq rhs$ muss daher in diesem Fall erfüllt sein.

Fall 2: Falls $h = 0$ gilt, darf die Hilfsvariable s beliebig große Werte annehmen, da sie nach oben unbeschränkt ist. Die Ungleichung $lhs \leq rhs + s$ ist somit in diesem Fall immer erfüllt.

Die verbleibende zweite Formel $h = 0 \implies \neg(lhs \leq rhs)$ formen wir als Nächstes zu $h' = 1 \implies rhs + \epsilon \leq lhs$ um. Wie zuvor kann nun auch die zweite Formel mithilfe von SOS1 im regulären Optimierungsproblem durch folgende Constraints ersetzt werden:

$$rhs + \epsilon \leq lhs + s' \text{ mit } \text{sos}(h', s') \quad \text{und} \quad h' + s' \geq \epsilon.$$

Erneut lassen sich zwei Fälle unterscheiden:

Fall 1: Falls $h' = 1$ gilt (und damit $h = 0$ wegen $h + h' = 1$), muss die Bedingung $s' = 0$ gelten, da höchstens ein Element des SOS1 $\text{sos}(h', s')$ größer als null sein darf. Die Ungleichung $rhs + \epsilon \leq lhs$ muss in diesem Fall erfüllt sein. Dies ist genau dann der Fall, wenn die Ungleichung des ursprünglichen Teil-Constraints $lhs \leq rhs$ nicht erfüllt ist.

Fall 2: Falls $h' = 0$ gilt, kann die Hilfsvariable s' beliebig negative Werte annehmen. Der Constraint $rhs + \epsilon \leq lhs + s'$ ist in diesem Fall immer erfüllt.

In diesem Abschnitt wurde die Transformation von Constraints der Form $h = 1 \Leftrightarrow lhs \leq rhs$ in Constraints des regulären Optimierungsproblems gezeigt. Die Umformung von Constraints $h = 1 \Leftrightarrow lhs \geq rhs$ des erweiterten Optimierungsproblems erfolgt analog zum gezeigten Vorgehen.

6.2.3 *Analyseziele*

Im Folgenden beschreiben wir die Zielfunktionen des regulären mathematischen Optimierungsproblems, um Clafer-Spezifikation auf die in Abschnitt 6.1.4 gezeigten semantischen Eigenschaften zu analysieren. Wir bezeichnen Zielfunktionen des Optimierungsproblems, die minimiert werden sollen, mit F_{min}, falls Clafer-Definitionen oder Multiplizitätsintervalle Analyseziel sind, und mit G_{min}, falls Gruppenkardinalitätsintervalle Analyseziel sind. Analog bezeichnen wir Zielfunktionen, die maximiert werden sollen, mit F_{max} bzw. G_{max}. Für jede Minimierung oder Maximierung ist mindestens ein Aufruf eines Standard-MILP-Solvers erforderlich. Dieser meldet potentiell folgende Ergebnisse:

✔ Falls eine minimale Lösung gefunden wird, existiert ein Zielfunktionswert F^*_{min} bzw. G^*_{min} für die Zielfunktion F_{min} bzw. G_{min}. Falls eine maximale Lösung gefunden wird, existiert ein Zielfunktionswert F^*_{max} bzw. G^*_{max} für die Zielfunktion F_{max} bzw. G_{max}.

✖ Falls keine Lösung gefunden wird, kann keine Zuweisung für Entscheidungsvariablen gefunden werden, die alle Constraints erfüllt.

✻ Falls der Solver die Unbeschränktheit des Suchraums meldet, existieren potentiell beliebig viele Lösungen. Es existieren zwar gültige Lösungen, diese sind jedoch nicht maximal, da der Suchraum beliebig ausgedehnt werden kann.

Im Weiteren betrachten wir zunächst die Formulierung der Konsistenzprüfung einer Clafer-Spezifikation als Zielfunktion des mathematischen Optimierungsproblems. Anschließend beschreiben wir die Formulierung der Analyse von Clafer-Definitionen, von Multiplizitäts- und Gruppenkardinalitätsintervallen sowie die Analyse von Clafer-Definitionen, die Attribute repräsentieren. Für alle Fälle interpretieren wir die genannten Ergebnisse, die ein Solver-Lauf liefern kann.

Konsistenzprüfung

In Abschnitt 6.1.4 wurde eine Spezifikation, die auf die erweiterte Kernsprache reduziert wurde, als konsistent charakterisiert, falls mindestens eine gültige Multimengen-Belegung existiert. Da per Definition (siehe Abschnitt 4.2.2) die Wurzel-Clafer-Definition c_r in jeder Instanz einer Spezifikation vorkommen muss, eignet sich die Prüfung auf das Vorhandensein einer Instanz der Wurzel-Clafer-Definition zur Analyse der Konsistenz der gesamten Spezifikation. Wir minimieren daher die Entscheidungsvariable $x_{c_r}^C$, welche die Anzahl der Instanzen der Wurzel-Clafer-Definition c_r angibt und in der Multimengen-Repräsentation durch den Zählfunktionswert $m_C(c_r)$ repräsentiert wird. Die Zielfunktion für die Konsistenzprüfung einer Clafer-Spezifikation ergibt sich somit zu

$$F_{min} = \arg\min_{X^C} x_{c_r}^C.$$

Die Interpretation der Solver-Ergebnisse für die Minimierung der Zielfunktion lautet wie folgt:

- ✔ Falls der Solver eine gültige Lösung findet, existiert mindestens eine gültige Spezifikationsinstanz, die alle Constraints erfüllt. In diesem Fall ist die Spezifikation konsistent. Falls die Spezifikation in der erweiterten Kernsprache konsistent ist, ist diese nicht zwangsläufig im ursprünglichen nicht-reduzierten Sprachumfang konsistent, da durch die Wegnahme von Constraints potentiell Inkonsistenzen beseitigt werden.
- ✖ Falls der Solver keine Lösung findet, ist die Spezifikation inkonsistent. Es kann somit keine Kombination von Entscheidungsvariablen gefunden werden, welche die Constraints erfüllt. Falls die Spezifikation in der erweiterten Kernsprache inkonsistent ist, so ist auch die ursprüngliche nicht-reduzierte Spezifikation inkonsistent (siehe Korrektheitseigenschaft des Analyseverfahrens in Abschnitt 4.3.4).
- ✻ Der Wertebereich für die Entscheidungsvariable $x_{c_r}^{*C}$ ist nach unten hin eingeschränkt, da negative Werte nicht zulässig sind. Für dieses Analyseziel kann der Suchraum daher nicht unbeschränkt bezüglich der Zielfunktion sein.

Analyse von Clafer-Definitionen

In Abschnitt 6.1.4 wurden Anomalien vom Typ *tote Clafer-Definition* und *Kern-Clafer-Definition* als semantische Eigenschaften von Clafer-Definitionen genannt. Diese lassen sich in der Multimengen-Repräsentation charakterisieren. Um Clafer-Definitionen einer Clafer-Spezifikation auf die genannten Eigenschaften zu analysieren, gehen wir folgendermaßen vor: Wir minimieren die Entscheidungsvariable x_c^C, um die Spezifikationsinstanz mit der minimal möglichen Anzahl von Instanzen der Clafer-Definition c zu identifizieren. Entsprechend maximieren wir die Entscheidungsvariable x_c^C, um die

maximal mögliche Anzahl von Instanzen der Clafer-Definition c in einer Clafer-Spezifikationsinstanz zu bestimmen. Die Zielfunktionen zur Analyse einer Clafer-Definition $c \in C \setminus (\{ c_r \} \cup C_{\mathbb{Z}} \cup C_{\mathbb{R}})$ lauten somit wie folgt:

$$F_{min} = \arg\min_{X^C} x_c^C \quad \text{und} \quad F_{max} = \arg\max_{X^C} x_c^C.$$

Solver-Aufrufe mit den genannten Zielfunktionen liefern Ergebnisse, die sich wie folgt interpretieren lassen:

✔ Falls der Solver eine optimale Lösung $F^*_{min} > 0$ ermittelt, existiert keine Clafer-Spezifikation, in der keine Instanzen von Clafer-Definition c vorkommt. In diesem Fall liegt somit eine Anomalie vom Typ *Kern-Clafer-Definition* vor. Eine Anomalie vom Typ *Kern-Clafer-Definition* liegt gemäß Definition 6.6 auch dann vor, falls durch die Hinzunahme weiterer Constraints, die nicht Bestandteil der erweiterten Kernsprache sind, oder durch die präzisere Betrachtung der Constraints, die Bestandteil der erweiterten Kernsprache sind, die Clafer-Definition eine Anomalie vom Typ *tote Clafer-Definition* besitzt. Falls der Solver eine optimale Lösung $F^*_{max} = 0$ ermittelt, existiert keine Clafer-Spezifikation, in der mindestens eine Instanz von Clafer-Definition c auftritt. Es liegt in diesem Fall somit eine Anomalie vom Typ *tote Clafer-Definition* vor.

  Die identifizierten Eigenschaften für eine auf die erweiterte Kernsprache reduzierte Clafer-Spezifikation existieren auch für die ursprüngliche Clafer-Spezifikation, da der Suchraum durch die Hinzunahme von Constraints gegenüber der erweiterten Kernsprache weiter eingeschränkt, jedoch nicht ausgedehnt wird (siehe Abschnitt 4.3.4).

✖ Falls der Solver keine gültige Lösung finden kann, existiert keine Instanz, in der Clafer-Definition c vorkommt oder nicht vorkommt. Die Clafer-Spezifikation ist somit inkonsistent, da keine gültige Instanz der Spezifikation existiert.

✻ Falls der Solver die Unbeschränktheit des Suchraums meldet, können Clafer-Spezifikationen mit potentiell beliebig vielen Instanzen der Clafer-Definition c gefunden werden. Die Clafer-Definition c und die Clafer-Spezifikation sind somit semantisch unbeschränkt. Dieses Resultat gilt jedoch nur für Clafer-Spezifikationen, die in der erweiterten Kernsprache vorliegen, da weggelassene Constraints den Suchraum beschränken könnten.

Analyse von Multiplizitäts- und Gruppenkardinalitätsintervallen

Die Analyse der Multiplizitätsintervalle $\lambda_c(c') = (l, u)$ bzw. Gruppenkardinalitätsintervalle $\lambda_g(c') = (l, u)$ einer Clafer-Definition $c' \in C \setminus (\{ c_r \} \cup C_{\mathbb{Z}} \cup C_{\mathbb{R}})$ dient der Detektion von Anomalien, die an Intervallgrenzen sowie innerhalb von Teilintervallen auftreten. Darüber hinaus dient die Analyse der Identifikation von Intervallen, die syntaktisch

unbeschränkt, jedoch semantisch beschränkt sind. Die Analyse des Multiplizitäts- bzw. Gruppenkardinalitätsintervalls einer Clafer-Definition c, die in c' geschachtelt ist (d. h. es gilt $c' \blacklozenge\!\!\rightarrow c$), erfolgt in zwei Stufen. In einem ersten Schritt bestimmen wir die minimale und von null verschiedene Anzahl k'_{min} der Eltern-Clafer-Definition c'. Dazu fügen wir den Constraint $x^C_{c'} \geq 1$ dem Optimierungsproblem hinzu und minimieren durch einen Solver-Aufruf die Zielfunktion $\arg\min_{X^C} x^C_{c'}$. Der Zielfunktionswert dieses Solver-Aufrufs entspricht k'_{min}.

In einem zweiten Schritt analysieren wir die Unter- und Obergrenze des Multiplizitätsintervalls $\lambda_c(c)$, indem wir die minimale bzw. maximale Anzahl von Instanzen der Clafer-Definition c für eine fixierte Anzahl von Instanzen k'_{min} der Eltern-Clafer-Definition c' bestimmen. Dazu fügen wir dem Optimierungsproblem den Constraint $x^C_{c'} = k'_{min}$ hinzu. Die Zielfunktionen F_{min} und F_{max} zur Analyse der Unter- und Obergrenze lauten folgendermaßen:

$$F_{min} = \arg\min_{X^C} x^C_c \quad \text{und} \quad F_{max} = \arg\max_{X^C} x^C_c.$$

Entsprechend analysieren wir die Unter- und Obergrenze eines Gruppenkardinalitätsintervalls λ_g, indem wir die Summe der Kinder-Clafer-Definitionen einer Eltern-Clafer-Definition c' für eine fixierte Anzahl von Instanzen der Eltern-Clafer-Definition minimieren bzw. maximieren. Erneut führen wir zum Fixieren der Anzahl der Eltern Clafer-Definition c' den Constraint $x^C_{c'} = k'_{min}$ ein. Die Zielfunktionen G_{min} und G_{max} sind wie folgt definiert:

$$G_{min} = \arg\min_{X^C} \sum_{c \in \{\, c \in C \mid c' \blacklozenge\!\!\rightarrow c \,\}} x^C_c \quad \text{und} \quad G_{max} = \arg\max_{X^C} \sum_{c \in \{\, c \in C \mid c' \blacklozenge\!\!\rightarrow c \,\}} x^C_c.$$

Für die Analyse der Gültigkeit spezifischer Werte $k'' \in \mathbb{N}_0$ eines Teilintervalls des Multiplizitätsintervalls fügen wir den zusätzlichen Constraint $x^C_c = k''$ ein und verwenden F_{max} als Zielfunktion. Für die Analyse der Gültigkeit spezifischer Werte eines Teilintervalls des Gruppenkardinalitätsintervalls fügen wir den zusätzlichen Constraint

$$\sum_{c \in \{\, c \in C \mid c' \blacklozenge\!\!\rightarrow c \,\}} x^C_c = k''$$

ein und verwenden G_{max} als Zielfunktion. Die Ergebnisse der Solver-Aufrufe für die genannten Zielfunktionen lassen sich wie folgt interpretieren:

- ✔ Falls wir die Untergrenze des Multiplizitäts- bzw. Gruppenkardinalitätsintervalls analysieren und der Solver den Zielfunktionswert F^*_{min} bzw. G^*_{min} liefert, können wir die Untergrenze des Multiplizitätsintervalls durch den Ausdruck $l^* = \lfloor F^*_{min} / k'_{min} \rfloor$ bzw. $l^* = \lfloor G^*_{min} / k'_{min} \rfloor$ bestimmen. Falls $l^* > l$ gilt, dann liegt eine Anomalie vom Typ *totes Kardinalitätsintervall* für das Intervall $[l, l^*]$ vor.

  Falls wir die Obergrenze des Multiplizitäts- bzw. Gruppenkardinalitätsintervalls analysieren und der Solver den Zielfunktionswert F^*_{max} bzw. G^*_{max} liefert, können wir

die Obergrenze des Intervalls durch den Ausdruck $u^* = \lfloor F^*_{max} / k'_{min} \rfloor + F^*_{max} \bmod k'$ bzw. $u^* = \lfloor G^*_{max} / k'_{min} \rfloor + G^*_{max} \bmod k'$ bestimmen. Falls $u^* < u$ gilt, liegt für das Teilintervall $[u^*, u]$ eine Anomalie vom Typ *totes Kardinalitätsintervall* vor. Falls $u = *$, dann ist das Multiplizitäts- bzw. Gruppenkardinalitätsintervall fälschlicherweise unbeschränkt.

Falls wir das Multiplizitäts- bzw. Gruppenkardinalitätsintervall bezüglich eines spezifischen Wertes k'' analysieren und der Solver eine (optimale) Lösung liefert, liegt für k'' keine Anomalie vor. Dieses Resultat muss jedoch für die ursprüngliche Spezifikation nicht gelten, da durch die Übersetzung in die erweiterte Kernsprache potentiell Constraints weggelassen wurden, die Anomalien für k'' einführen.

✖ Falls wir die Ober- und Untergrenzen des Multiplizitätsintervalls analysieren und der Solver keine Lösung identifiziert, liegt eine Anomalie vom Typ *tote Clafer-Definition* vor. Entsprechendes gilt, wenn wir die Ober- und Untergrenze des Gruppenkardinalitätsintervalls analysieren und der Solver keine Lösung findet. In diesem Fall existiert keine Kombination der Kind-Clafer-Definitionen, welche die Einschränkungen des Gruppenkardinalitätsintervalls erfüllen. Falls wir einen spezifischen Wert k'' des Intervalls analysieren und der Solver keine Lösung identifiziert, existiert keine Kombination von Instanzen der Kind-Clafer-Definitionen, die in Summe den Wert k'' ergibt. Es liegt somit für k'' eine Anomalie vom Typ *totes Kardinalitätsintervall* vor.

Detektierte Anomalien für eine auf die erweiterte Kernsprache reduzierte Clafer-Spezifikation existieren auch für die vollständige Clafer-Spezifikation, da durch die Wegnahme von Constraints der Suchraum ausgedehnt, jedoch nicht weiter eingeschränkt wird (siehe Abschnitt 4.3.4).

✻ Die Lösungsmenge kann nur bei der Analyse der Obergrenze unbeschränkt sein, da bei der Analyse der Untergrenze und spezifischer Werte der Suchraum durch Constraints beschränkt ist. Falls der Solver die Unbeschränktheit der Lösungsmenge meldet, ist das Multiplizitätsintervall λ_c bzw. das Gruppenkardinalitätsintervall λ_g tatsächlich unbeschränkt. Falls ein Intervall in der erweiterten Kernsprache unbeschränkt ist, so ist dies nicht zwangsläufig für die vollständige Clafer-Spezifikation der Fall, da Constraints für die Transformation in die erweiterte Kernsprache potentiell verworfen wurden (siehe Abschnitt 4.3.4).

Analyse von Attributen

Wie in Abschnitt 4.1 gezeigt, werden Attribute in der Sprache Clafer durch Referenzen $c \twoheadrightarrow c_z$ repräsentiert. Dabei ist c_z eine Hilfe-Clafer-Definition mit $c_z \in C_{\mathbb{Z}}$, falls c die Menge **`integer`** referenziert (und somit ein ganzzahliges Attribut repräsentiert). Es gilt $c_z \in C_{\mathbb{R}}$, falls c die Menge **`real`** referenziert (und somit ein reellwertiges Attribut

repräsentiert). Zur Transformation der genannten Referenz in die Multimengen-Repräsentation wurde der Clafer-Definition c in Abschnitt 5.3 durch die Funktion numRef(c) eine Variable $v \in V_{\mathbb{Z}}$ bzw. $v \in V_{\mathbb{R}}$ zugeordnet, die einen minimalen oder maximalen Wert der Instanzmenge der Clafer-Definition c_z repräsentiert. In Abschnitt 6.2.1 wurde jeder Variablen v eine ganzzahlige Entscheidungsvariable $x_v^{V_{\mathbb{Z}}}$ bzw. eine reellwertige Entscheidungsvariable $x_v^{V_{\mathbb{R}}}$ zugeordnet. Zur Ermittlung einer gültigen minimalen und maximalen Belegung der Variablen v minimieren oder maximieren wir daher die Entscheidungsvariable x_v^V mit $V = V_{\mathbb{Z}} \cup V_{\mathbb{R}}$. Die Zielfunktionen F_{min} und F_{max} ergeben sich somit wie folgt:

$$F_{min} = \arg\min_V x_v^V \quad \text{und} \quad F_{max} = \arg\max_V x_v^V.$$

✔ Falls der Solver eine minimale oder maximale Lösung mit dem Zielfunktionswert F_{min}^* oder F_{max}^* liefert, entspricht der Zielfunktionswert dem minimal bzw. maximal möglichen ganzzahligen oder reellen Wert einer Instanz der Clafer-Definition c_z. Die Clafer-Definition c_z ist somit durch einen Constraint auf einen Wertebereich eingeschränkt. Ein ungültiger Wertebereich, der für eine auf die Kernsprache reduzierte Spezifikation ermittelt wurde, ist auch für die nicht-reduzierte Spezifikation ungültig, da er durch zusätzliche Constraints weiter eingeschränkt jedoch nicht ausgedehnt wird.

✖ Falls der Solver keine gültige Lösung liefert, existiert keine Multimengen-Belegung für beliebige Werte von v. Die Spezifikation ist somit inkonsistent. Ergebnisse dieser Analyse für Spezifikationen, die in der erweiterten Kernsprache vorliegen, sind aus dem oben genannten Grund übertragbar auf Spezifikationen, die den kompletten Sprachumfang von Clafer nutzen.

✻ Falls der Solver meldet, dass der Suchraum für die Zielfunktion nach oben oder nach unten unbeschränkt ist, kann für die Entscheidungsvariable x_v^V potentiell ein beliebig großer negativer oder positiver Wert gewählt werden, ohne die Constraints des Optimierungsproblems zu verletzen. Der Wertebereich des durch die Clafer-Definition c repräsentierten Attributs ist somit nach unten bzw. oben unbeschränkt. Ergebnisse dieser Analyse für Spezifikationen, die in der erweiterten Kernsprache vorliegen, gelten nicht zwangsläufig für nicht-reduzierte Spezifikationen, da der Wertebereich durch weggelassene Constraints potentiell erweitert wird.

7

EXPERIMENTELLE EVALUATION

In diesem Kapitel untersuchen wir das zuvor vorgestellte Verfahren zur Analyse von Clafer-Spezifikationen im Rahmen einer experimentellen Evaluation. Dabei verfolgen wir zwei Zielsetzungen:

A Das erste Evaluationsziel ist die *Untersuchung der Effektivität* unseres Analyseverfahrens bezüglich Korrektheit und Relevanz der identifizierten Eigenschaften für repräsentative Clafer-Spezifikationen, die in Größe und Komplexität variieren.

B Das zweite Evaluationsziel ist die *Untersuchung der Effizienz* unseres Analyseverfahrens in Bezug auf die Laufzeit der Analyse im Vergleich zu existierenden Ansätzen und bezüglich der Größe der Laufzeitrepräsentation, die sich potentiell durch die in Abschnitt 5.1 gezeigte Abflachung der Vererbungshierarchie und der in Abschnitt 5.2 gezeigten Multimengen-Repräsentation ergeben.

Zunächst werden in Abschnitt 7.1 Forschungsfragen identifiziert, die durch die experimentelle Evaluation beantwortet werden. Darauf aufbauend wird in Abschnitt 7.2 die Implementierung unseres Analyseverfahrens vorgestellt. Ausgehend von den Forschungsfragen werden in Abschnitt 7.3 die Korrektheit und in Abschnitt 7.4 die Anwendbarkeit unseres Analyseverfahrens untersucht (Evaluationsziels A). In den Abschnitten 7.5 und 7.6 folgt eine quantitative Untersuchung der Einflüsse unseres Analyseverfahrens auf Modellgrößen sowie Rechenaufwand (Evaluationsziels B).

7.1 FORSCHUNGSFRAGEN

Wir berücksichtigen vier Forschungsfragen, die in zwei Gruppen entsprechend der Evaluationsziele unterteilt sind. Zur Erreichung des Evaluationsziels A untersuchen wir die Effektivität unseres Analyseverfahrens in Bezug auf die Korrektheit und Anwendbarkeit auf praxisrelevante Clafer-Spezifikationen. Es ergeben sich somit folgende Forschungsfragen für Evaluationsziel A:

- FF A.1 (Korrektheit): Korrespondieren Anomalien, die durch unser Analyseverfahren für Clafer-Spezifikationen in der Kernsprache detektiert werden, mit Anomalien in Clafer-Spezifikationen, die nicht in die Kernsprache transformiert wurden?

- FF A.2 (Anwendbarkeit): Wie viele Anomalien können in praxisrelevanten Clafer-Spezifikationen durch unser Verfahren detektiert werden und wie viele dieser Anomalien können nicht durch existierende Verfahren identifiziert werden?

Für unser Analyseverfahren ist es notwendig, dass eine Clafer-Spezifikation durch mehrere Transformationsschritte zur erweiterten Kernsprache reduziert und in unsere Multimengen-Repräsentation transformiert wird. Bezüglich des Evaluationsziels B untersuchen wir daher, welchen Einfluss die Transformationsschritte auf die Modellgrößen der Zwischenrepräsentationen, die hierdurch resultierende Komplexität des mathematischen Optimierungsproblems und deren potentiellen Einfluss auf die Laufzeiteffizienz in Form von CPU-Zeit haben. In dieser Hinsicht vergleichen wir die Laufzeit unseres Analyseverfahrens mit existierenden Ansätzen. Es ergeben sich somit folgende Forschungsfragen für Evaluationsziel B:

- FF B.1 (Modellgrößen): Wie beeinflussen die Schritte zur Transformation einer Clafer-Spezifikation in die Kernsprache die Modellgrößen der Zwischenrepräsentationen? Was ist die Komplexität des hieraus resultierenden mathematischen Optimierungsproblems?
- FF B.2 (Rechenaufwand): Welchen Einfluss hat unser Analyseverfahren auf den Rechenaufwand für die Konsistenzprüfung im Vergleich zu existierenden Ansätzen zur Instanzgenerierung von Clafer-Spezifikationen?

Die Abschnitte 7.3 bis 7.6 beantworten die Forschungsfragen. Für jede Forschungsfrage stellen wir den Versuchsaufbau dar und beschreiben sowie diskutieren die Ergebnisse.

7.2 IMPLEMENTIERUNG

Das im Rahmen dieser Arbeit vorgestellte Analyseverfahren zur Konsistenzprüfung und Anomaliedetektion von Clafer-Spezifikation haben wir in dem Software-Tool BOUNDANALYZER implementiert[1]. Die Architektur orientiert sich an dem in [3] vorgeschlagenen Aufbau von Compilern: Wir unterscheiden zwischen Analyse-Komponenten und Synthese-Komponenten. Analyse-Komponenten prüfen die syntaktische und statische Korrektheit der zu analysierenden Clafer-Spezifikation. Synthese-Komponenten erzeugen die in Kapitel 5 vorgestellte Multimengen-Repräsentation als Zwischenrepräsentation und generieren nach dem in Kapitel 6 gezeigten Vorgehen ein mathematisches Optimierungsproblem passend zum gewählten Analyseziel. Ein Überblick über die Architektur von BOUNDANALYZER zeigt Abbildung 7.1. Im Folgenden stellen wir die Analyse- sowie Synthese-Komponenten von BOUNDANALYZER vor.

1 https://github.com/mtweck/cardygan-boundanalyzer

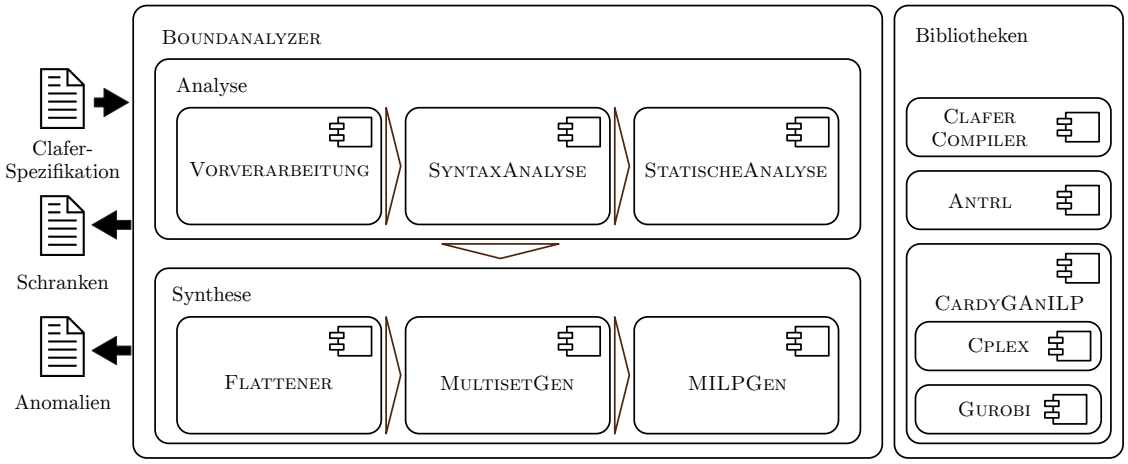


Abbildung 7.1: Überblick über Architektur von BOUNDANALYZER

Analyse-Komponenten

Als Eingabe der Analyse-Komponenten werden Clafer-Spezifikationen akzeptiert, die den vollständigen Sprachumfang der Sprache Clafer enthalten dürfen. Durch programmatische Schnittstellen besteht zudem die Möglichkeit, BOUNDANALYZER in andere Software einzubinden. So wird beispielsweise BOUNDANALYZER in der Toolsuite CARDYGAN verwendet, um Anomalien in kardinalitätsbasierten Feature-Modellen zu detektieren [134]. Die Komponente VORVERARBEITUNG führt die Vorverarbeitung von Clafer-Spezifikationen durch. Bei der Vorverarbeitung werden Clafer-Spezifikationen normalisiert, indem Syntaxerweiterungen beseitigt werden, die der Vereinfachung der Schreibweise dienen. Der Vorverarbeitungsschritt erfolgt mithilfe des Clafer-Compilers in Version 0.4.3, der Referenzimplementierung der Sprachspezifikation von Clafer. Die Komponente SYNTAXANALYSE führt anschließend die lexikalische und syntaktische Analyse basierend auf dem Parser-Generator ANTRL [109] in Version 4.6 durch. Die Komponente STATISCHEANALYSE erzeugt zunächst einen Abhängigkeitsgraphen der Clafer-Spezifikation (siehe Abschnitt 4.2). Darauf aufbauend werden Wohlgeformtheitseigenschaften geprüft und die Reduktion der Clafer-Spezifikation auf die erweiterte Kernsprache durchgeführt. Dies erfolgt, indem Constraints verworfen werden, welche die in Abschnitt 6.1 gezeigten Kriterien der erweiterten Kernsprache nicht erfüllen.

Synthese-Komponenten

Die Synthese-Komponenten umfassen die Teil-Komponenten FLATTENER, MULTISETGEN und MILPGEN. Die Komponente FLATTENER implementiert das in Abschnitt 5.1 gezeigte Vorgehen zur Abflachung der Vererbungshierarchie und liefert für die folgenden Verarbeitungsschritte eine abgeflachte Clafer-Spezifikation. Die Komponente MULTISETGEN implementiert das in Kapitel 5 gezeigte Vorgehen zur Transformation einer abgeflachten

Clafer-Spezifikation in eine Multimengen-Repräsentation, die als Zwischenrepräsentation dient. Die Komponente MILPGen generiert gemäß dem in Abschnitt 6.2 gezeigten Vorgehen aus der Multimengen-Repräsentation dem jeweiligen Analyseziel entsprechend ein erweitertes gemischt-ganzzahliges Optimierungsproblem (MILP). Die Zielrepräsentation ist hierbei eine Solver-unabhängige Sprache, die durch die Komponente CardyGAnILP zur Verfügung gestellt wird. Die Komponente CardyGAnILP implementiert das in Abschnitt 6.2.2 gezeigte Vorgehen zur Transformation von erweiterten mathematischen Optimierungsproblemen (die neben Constraints, bestehend aus linearen Ungleichungen, auch die aussagenlogische Verknüpfung von Constraints zulassen) in reguläre gemischt-ganzzahlige Optimierungsprobleme.Um das zugrundeliegende Optimierungsproblem zu lösen, verwenden wir den MILP-Solver Gurobi [63]. Die Ergebnisse der Solver-Aufrufe können entweder programmatisch durch eine Programmierschnittstelle oder als Dateiausgabe abgerufen werden.

7.3 UNTERSUCHUNG DER KORREKTHEIT

In diesem Abschnitt untersuchen wir die Korrektheit unseres Analyseverfahrens (FF A.1). Dazu prüfen wir ob Anomalien, die durch unser Analyseverfahren für Clafer-Spezifikationen in der Kernsprache detektiert wurden, mit Anomalien in Clafer-Spezifikationen übereinstimmen, die nicht zur Kernsprache transformiert wurden. In Abschnitt 4.3.4 wurde Korrektheit als wesentliche Anforderung an unser Analyseverfahren beschrieben. Im Folgenden beschreiben wir zunächst den Versuchsaufbau zur Untersuchung der Korrektheit unseres Analyseverfahrens. Im Anschluss beschreiben und diskutieren wir die Ergebnisse der Untersuchung.

7.3.1 *Versuchsaufbau*

Die Überprüfung der Korrektheit unseres Analyseverfahrens ist ein Spezialfall des sehr schwierigen Problems des Nachweises der Korrektheit von Compilern oder Modelltransformatoren [37, 120]. Den Nachweis der Korrektheit unseres Analyseverfahrens sichern wir deshalb zusätzlich durch dynamische Validierung unseres Analyseverfahrens ab. Eine wesentliche Herausforderung stellt hierbei das Testorakel-Problem [14] dar: Für einen gegebenen Programmcode als Testeingabe ist es sehr schwer, die erwartete Ausgabe des Compilers festzustellen.

Ein etabliertes Vorgehen, um dieses Problem anzugehen, stellt Randomized Differiantial Testing (DRT) [101, 155] dar. Hierbei werden die Ausgaben verschiedener Compiler-Implementierungen für eine Testeingabe verglichen. Die Compiler-Implementierungen basieren auf der gleichen Sprachspezifikation. Falls die Compiler unterschiedliche Ausgaben für die gleiche Eingabe erzeugen, kann daraus geschlossen werden, dass einige Compiler-Implementierungen möglicherweise Programmfehler aufweisen.

Um die Korrektheit unseres Analyseverfahrens zu überprüfen, wenden wir ein ähnliches Vorgehen an: Für eine repräsentative Menge von Clafer-Spezifikationen vergleichen wir erwartete Anomalien und Inkonsistenzen mit den tatsächlichen Ergebnissen unseres Analyseverfahrens. Die erwarteten Anomalien und Inkonsistenzen einer Clafer-Spezifikation bestimmen wir durch bestehende Implementierungen zur Generierung von Clafer-Spezifikationsinstanzen. Zum Zeitpunkt des Verfassens dieser Arbeit sind zwei Implementierungen zur Generierung von Clafer-Spezifikationsinstanzen verfügbar.

- *Alloy-basierte Implementierung*: Eine Clafer-Spezifikation wird hierbei in die strukturelle Modellierungssprache Alloy [76] übersetzt. Das gleichnamige Software-Werkzeug ALLOY stellt einen SAT-basierten Instanzgenerator bereit, der innerhalb eines beschränkten Suchraums nach Instanzen sucht. Der Suchraum wird beschränkt, indem Obergrenzen für die Anzahl von Clafer-Definitionen angegeben werden. Wir verwenden für die Experimente ALLOY in der Version 4.2 mit dem SAT-Solver MINISAT [49].

- *Constraint-Programming-basierte Implementierung*: Eine Clafer-Spezifikation wird hierbei als Constraint-Programming-Problem codiert, für das mithilfe eines Constraint-Programming-Solvers [115] eine Lösung gesucht wird. Auch hier muss der Suchraum eingeschränkt werden, indem Obergrenzen für die Anzahl von Clafer-Definitionen angegeben werden. Wir verwenden für die Experimente das Clafer Constraint-Programming-Backend CHOCOSOLVER in der Version 4.3 [6].

Beide Implementierungen verwenden wir in Kombination mit dem CLAFERCOMPILER in Version 4.3. Zum Zeitpunkt des Verfassens dieser Arbeit können durch die genannten Verfahren keine Clafer-Spezifikation mit reellwertigen Attributen (Clafer-Definitionen, welche die primitive Menge `real` referenzieren) behandelt werden. Sonstige konzeptionelle Einschränkungen des Umfangs der Sprache Clafer bei der Instanzgenerierung bestehen darüber hinaus nicht.

Wie in Abschnitt 4.3.4 gezeigt, können wir für eine Clafer-Spezifikation durch einen zusätzlichen Constraint erzwingen, dass alle Clafer-Spezifikationsinstanzen über die gleiche Anzahl von Instanzen einer Clafer-Definition verfügen. Existierende Verfahren für die Generierung von Clafer-Spezifikationsinstanzen können auf diese Weise eingesetzt werden, um für eine geforderte Anzahl an Instanzen einer Clafer-Definition zu prüfen, ob Clafer-Spezifikationsinstanzen existieren. Falls keine Instanz für eine geforderte Anzahl einer Clafer-Definition gefunden wird, obwohl dies beispielsweise durch ein Multiplizitätsintervall als zulässig spezifiziert wurde, liegt für die geprüfte Anzahl eine Anomalie oder Inkonsistenz vor.

Zur Überprüfung der Korrektheit unseres Analyseverfahrens gehen wir abhängig vom Analyseziel wie folgt vor:

- Falls wir die Korrektheit der Analyse von Clafer-Definitionen und Attributen prüfen, bestimmen wir entsprechend des in Abschnitt 6.2.3 gezeigten Vorgehens die

minimale und maximale Anzahl von Instanzen der Clafer-Definition $c \in C$ bzw. der Clafer-Definition $c \in C_{\mathbb{Z}}$, die ein ganzzahliges Attribut repräsentiert. Die berechnete minimale und maximale Anzahl entspricht den Werten l^* und u^*. Falls $l^* > 0$ gilt, fügen wir in einem zweiten Schritt der obersten Ebene der Schachtelungshierarchie einen zusätzlichen Constraint der Form `#c` $< l^*$ hinzu. Entsprechend, falls $u^* \neq *$ gilt, fügen wir einen Constraint der Form `#c` $> u^*$ hinzu.

- Falls wir die Korrektheit der Analyse von Multiplizitätsintervallen prüfen, bestimmen wir entsprechend des in Abschnitt 6.2.3 gezeigten Vorgehens die minimale Untergrenze l^* und maximale Obergrenze u^* des Multiplizitätsintervalls $\lambda_c(c) = (l, u)$ einer Clafer-Definition c. Falls $l < l^*$ gilt, fügen wir einen Constraint der Form `#(this.c)` $< l^*$ auf der Ebene der Eltern-Clafer-Definition von c der Schachtelungshierarchie hinzu. Entsprechend, falls $u^* < u$ gilt, fügen wir einen Constraint der Form `#(this.c)` $> u^*$ hinzu.

- Falls wir die Korrektheit der Analyse von Gruppenkardinalitätsintervallen prüfen, bestimmen wir zunächst entsprechend des in Abschnitt 6.2.3 gezeigten Vorgehens die minimale Untergrenze l^* und maximale Obergrenze u^* des Gruppenkardinalitätsintervalls $\lambda_g(c) = (l, u)$ einer Clafer-Definition c. Falls $l < l^*$ gilt, fügen wir einen Constraint der Form `#(this.c + this.c' + ...)` $< l^*$ auf der Ebene einer Clafer-Definition hinzu, die Eltern-Clafer-Definition der in dem Constraint addierten Clafer-Definitionen $c, c', \ldots$ ist. Entsprechend fügen wir, falls $u^* < u$ gilt, einen Constraint der Form `#(this.c + this.c' + ...)` $> u^*$ hinzu.

Für jede auf die genannte Weise modifizierte Clafer-Spezifikation verwenden wir die Alloy- und Constraint-Programming-basierten Implementierungen, um nach Spezifikationsinstanzen zu suchen. Hierfür ist es notwendig, den Suchraum je Clafer-Definition einzuschränken. Für die beschriebenen Experimente limitieren wir die Anzahl von Instanzen je Clafer-Definition c auf Werte $u^* + 10$, wobei u^* jeweils der maximalen Anzahl einer Clafer-Definition in einer Clafer-Spezifikation entspricht, die durch unser Analyseverfahren berechnet wurde. Falls unser Analyseverfahren die Unbeschränktheit einer Clafer-Definition feststellt, ermitteln wir eine plausible Oberschranke durch manuelle Inspektion der jeweiligen Clafer-Spezifikation. Unser Analyseverfahren funktioniert korrekt für eine Clafer-Spezifikation und eine Clafer-Definition, falls die existierenden Implementierungen für alle nach dem gezeigten Vorgehen modifizierten Spezifikationen ebenfalls keine Clafer-Spezifikationsinstanzen finden.

Als Testeingaben greifen wir auf eine etablierte Testsuite zurück, die zum Testen des Clafer-Compilers verwendet wird. Die Compiler-Testsuite enthält 81 Spezifikationen, die eine große Bandbreite von Elementen der Sprache Clafer abdecken. Zusätzlich berücksichtigen wir eine eigene Testsuite bestehend aus 60 Spezifikationen.

Tabelle 7.1 zeigt einen Überblick über die Arten und Häufigkeit der Syntax-Elemente der untersuchten Spezifikationen. Es ist zusätzlich die Häufigkeit von Pfadausdrücken angegeben, die keines oder bis zu vier **`dref`**-Schlüsselwörter enthalten.

Syntax-Element	Anzahl	Syntax-Element	Anzahl
Clafer-Def.	1341	`no sExp`	26
absrakte Clafer-Def.	199	`sExp in sExp'`	25
Constraints	245	`sExp not in sExp'`	2
Referenzen	126	`sExp = sExp'`	36
Numerische Referenzen	77	`sExp != sExp'`	5
`! bExp`	2	`sExp ++ sExp'`	5
`bExp && bExp'`	19	`sExp -- sExp'`	1
`bExp || bExp'`	4	`sExp ** sExp'`	2
`bExp => bExp'`	40	`# sExp`	58
`nExp > nExp'`	10	`nExp + nExp'`	4
`nExp >= nExp'`	18	`- nExp`	15
`nExp < nExp'`	16	`nExp * nExp'`	2
`nExp <= nExp'`	2	`pExpr` ohne `dref`	328
`nExp = nExp'`	198	`pExpr` mit #`dref` = 1	244
`lone sExp`	1	`pExpr` mit #`dref` = 2	48
`one sExp`	4	`pExpr` mit #`dref` = 3	26
`some sExp`	174	`pExpr` mit #`dref` ≥ 4	1
`no sExp`	26		

Tabelle 7.1: Anzahl der Syntax-Elemente der untersuchten Spezifikationen

	Alloy	Chocosolver
Offizielle Testsuite	100 % (2523/2523)	100 % (2523/2523)
Erweiterte Testsuite	100 % (999/999)	80 % (802/999)

Tabelle 7.2: Übereinstimmungen nach Ausführung der Testergebnisse

7.3.2 *Ergebnisse*

Tabelle 7.2 zeigt die Ergebnisse der Untersuchung der Korrektheit unseres Analyseverfahrens. Insgesamt wurden für Spezifikationen der offiziellen Testsuite 2532 Testfälle ausgeführt. Je Testfall wird entweder (i) die minimale und maximale Anzahl von Instanzen einer Clafer-Definition, (ii) die Intervallgrenzen des Multiplizitätsintervalls einer Clafer-Definition, (iii) die Intervallgrenzen des Gruppenkardinalitätsintervalls einer Clafer-Definition oder (iv) die Unter- und Obergrenze des Wertebereichs einer Clafer-Definition, die ein Attribut repräsentiert, analysiert. Für Spezifikationen der offiziellen Testsuite wurden in allen Fällen die identifizierten Anomalien durch Alloy und Chocosolver bestätigt: In allen Fällen konnten für entsprechend präparierte Spezifikationen keine Instanzen identifiziert werden.

Für Spezifikationen der erweiterten Testsuite konnten durch Alloy alle von unserem Analyseverfahren identifizierten Anomalien bestätigt werden. Im Falle von Chocosolver konnte nur in 80 % der Testfälle keine Instanz für entsprechend präparierte Spezifikationen gefunden werden. In 20 % der Fälle kam es zu reproduzierbaren Abstürzen des Chocosolver. In keinem Testfall konnte durch Alloy oder Chocosolver für Bereiche von Intervallen von Wertebereichen eine Spezifikationsinstanz identifiziert werden, die von unserem Analyseverfahren als ungültig eingestuft wurde.

Tabelle 7.3 zeigt die Messungen der Abdeckung des Programmcodes der Implementierung unseres Analyseverfahrens durch die ausgeführten Testfälle. Die Ergebnisse sind je Komponente aufgeführt und unterscheiden Anweisungs- und Zweigüberdeckung. Für die Komponenten StatischeAnalyse und Flattener konnten 93 % bzw. 94 % Anweisungsüberdeckung und 82 % bzw. 83 % Zweigüberdeckung erzielt werden. Für die Komponenten MILPGen und MILPGen wurden 87 % und 89 % Anweisungsüberdeckung und jeweils 73 % Zweigüberdeckung erzielt.

7.3.3 *Diskussion*

Die Forschungsfrage FF A.1 zur Untersuchung der Korrektheit unseres Analyseverfahrens beantworten wir wie folgt: Für alle Testfälle der verwendeten Testsuiten konnte die Korrektheit unseres Analyseverfahrens gezeigt werden. In 197 Fällen divergieren die Ergebnisse für Alloy und Chocosolver aufgrund von Programmabstürzen des

Komponente	Anweisungsüberdeckung	Zweigüberdeckung
StatischeAnalyse	93 %	82 %
Flattener	94 %	83 %
MultisetGen	87 %	73 %
MILPGen	89 %	73 %

Tabelle 7.3: Überdeckungsmaße je Komponente

Werkzeugs Chocosolver. Durch die Testfälle konnte ein Großteil der Anweisungen des Programmcodes unseres Analyseverfahrens abgedeckt werden (mindestens 89 %). Zusätzlich wurden durch die Testfälle ein Großteil der Zweige des Kontrollflusses des Programmcodes unseres Analyseverfahrens überdeckt (mindestens 73 %).

Da wir bei der Untersuchung der Korrektheit unseres Analyseverfahrens ausschließlich syntaktisch korrekte und semantisch sinnvollen Spezifikationen betrachten, wird durch die Testfälle kein Programmcode zur Ausführung gebracht, der Fehlerbehandlung oder aufwändige Plausibilitätsprüfungen enthält. Durch manuelle Inspektion des Programmcodes sowie der Testresultate konnten wir feststellen, dass sich hierdurch die vergleichsweise geringe Zweigüberdeckung für die Komponenten MultisetGen und MILPGen erklären lassen. Die Funktionalität der Fehlerbehandlung und Plausibilitätsprüfungen haben wir daher durch zusätzliche Negativtests abgesichert.

7.4 UNTERSUCHUNG DER ANWENDBARKEIT

In diesem Abschnitt untersuchen wir die Anwendbarkeit unseres Analyseverfahrens (FF A.2). Wir bestimmen dazu die Typen und Anzahl von Anomalien und Inkonsistenzen in praxisrelevanten Clafer-Spezifikationen, die durch unser Analyseverfahren detektiert werden. Darüber hinaus untersuchen wir, welche Anomalietypen durch unser Analyseverfahren, jedoch nicht durch existierende Verfahren, identifiziert werden.

7.4.1 *Versuchsaufbau*

Wir wenden unsere Experimente auf sieben repräsentative Systeme an, die in Größe und Komplexität variieren. Eines der untersuchten Systeme ist das in Abschnitt 2.1 vorgestellte selbst-adaptive Kommunikationssystem, dessen Clafer-Spezifikation in Listing 3.1 zu sehen ist. Bei den anderen Systemen handelt es sich um Clafer-Spezifikationen existierender Fallstudien der Sprache Clafer [10, 127, 148]. Alle betrachteten Spezifikationen sind syntaktisch korrekt und decken einen Großteil der verfügbaren Sprachkonstrukte ab, die durch die existierenden Implementierungen der Instanzgeneratoren unterstützt

werden. Außer der im Rahmen dieser Arbeit betrachteten Fallstudie, beinhaltet keine betrachtete Clafer-Spezifikation reellwertige Attribute. Tabelle 7.4 liefert eine Übersicht über die untersuchten Systeme und deren Charakteristika in Bezug auf die verwendeten Sprachelemente.

System	# Clafer-Definitionen	# Abs. Clafer-Definitionen	# Constraints	# Integer Referenzen	# Mengenreferenzen	# Synt. unbeschränkte λ_c	# Synt. unbeschränkte λ_g	Beschreibung
BC	185	33	225	64	19	32	182	Domänenmodell einer E/E-Architektur eines Zentralverriegelungssystems[2] [127]
BMM	156	24	19	4	35	12	141	Schema und Struktur eines generischen Business Motivation Models (BMM)[3]
EDM	41	5	14	2	12	8	41	Domänenmodell eines Raumplanungssystems[3]
PerR	13	1	12	1	3	2	12	Domänenmodell eines Familienstammbaums[3]
SAS	34	3	27	1	6	13	31	Integrierte Produktlinien-Spezifikation eines selbst-adaptiven Kommunikationssystems (siehe Listing 3.1)
TM	34	0	10	6	3	0	30	Domänenmodell eines Telematik-Systems[3] [10]
TnT	22	3	12	1	2	8	18	Integrierte Produktlinien-Spezifikation eines Track-and-Trace-Systems [148]

Tabelle 7.4: Betrachtete Systeme

Um die Anwendbarkeit unseres Analyseverfahrens zu untersuchen, analysieren wir Clafer-Definitionen, Multiplizitätsintervalle λ_c von Clafer-Definitionen, Gruppenkardinalitätsintervalle λ_g und Wertebereiche von Clafer-Definitionen, die Attribute repräsentieren, auf Anomalien. Bezüglich Clafer-Definitionen berechnen wir die Anzahl der Anomalien vom Typ *tote Clafer-Definition* und die Anzahl semantisch unbeschränkter Clafer-Definitionen. Bezüglich der Multiplizitäts- und Gruppenkardinalitätsintervalle bestimmen wir die Anzahl der Intervalle mit Anomalien vom Typ *falscher Untergrenze* und *falsche Intervall Obergrenze*. Zusätzlich bestimmen wir die Anzahl von Intervallen, die eine Anomalie vom Typ *fälschlicherweise unbeschränkt* aufweisen. Bezüglich Clafer-

2 https://github.com/gsdlab/ClaferCaseStudies
3 https://github.com/gsdlab/clafer/tree/master/test/positive

Definitionen, die Attribute repräsentieren, bestimmen wir die Anzahl von Attribute, deren Wertebereich nach unten bzw. oben hin beschränkt sowie unbeschränkt ist.

7.4.2 *Ergebnisse*

Die Ergebnisse der Versuchsdurchführung sind in Tabelle 7.5 zu sehen. Für fast alle untersuchten Systeme konnten Anomalien identifiziert werden. Bezüglich der Multiplizitätsintervalle von Clafer-Definitionen konnten insgesamt 28 Anomalien vom Typ *tote Clafer-Definition*, 6 vom Typ falsche Intervalluntergrenze und 48 Anomalien vom Typ *falsche Intervallobergrenze* durch unser Analyseverfahren identifiziert werden. Insgesamt konnten 32 fälschlicherweise unbeschränkte Multiplizitätsintervalle durch unser Analyseverfahren entdeckt werden. Zusätzlich konnten insgesamt 18 Multiplizitätsintervalle identifiziert werden, die in der erweiterten Kernsprache tatsächlich unbeschränkt sind. Für das in dieser Arbeit vorgestellte selbst-adaptive Kommunikationssystem wurden hierbei 10 und für das industrielle Track-and-Trace-System wurden 8 tatsächlich unbeschränkte Multiplizitätsintervalle identifiziert.

Bezüglich der Gruppenkardinalitätsintervalle konnten insgesamt 50 Anomalien vom Typ *falsche Intervalluntergrenze* und 87 Anomalien vom Typ *falsche Intervallobergrenze* durch unser Analyseverfahren identifiziert werden. Für fast alle Systeme konnten Gruppenkardinalitätsintervalle identifiziert werden, die in der erweiterten Kernsprache fälschlicherweise unbeschränkt sind (insgesamt 87).

Bezüglich der Analyse der Wertebereiche von Attributen konnte durch unser Analyseverfahren für 28 Wertebereiche erkannt werden, dass diese nach oben und unten beschränkt sind. Zusätzlich wurden durch unser Analyseverfahren 136 Wertebereiche von Attributen identifiziert, die in der erweiterten Kernsprache unbeschränkt sind.

7.4.3 *Diskussion*

Die Forschungsfrage FF A.2 zur Untersuchung der Anwendbarkeit unseres Analyseverfahrens beantworten wir wie folgt: Die Ergebnisse zeigen eine vergleichsweise große Anzahl von Anomalien, die in den Spezifikationen der untersuchten Systeme durch unser Analyseverfahren identifiziert werden konnten. Diese können auf mögliche Mängel in der Spezifikation hinweisen.

Die sehr große Anzahl der identifizierten fälschlicherweise unbeschränkten Gruppenkardinalitätsintervalle lässt sich wie folgt erklären: Falls kein Gruppenkardinalitätsintervall für eine Clafer-Definition explizit spezifiziert wird, wird standardmäßig ein Gruppenkardinalitätsintervall der Form `0..*` angenommen. Unser Analyseverfahren identifiziert hingegen für alle Gruppenkardinalitätsintervalle, soweit möglich, obere Intervallgrenzen. In insgesamt 87 Fällen können die unbeschränkten oberen Grenzen durch konkrete Werte ersetzt und so die Aussagekraft der Spezifikation der untersuchten Systeme erhöht werden.

	λ_c					λ_g			Attribut		
System	**# tote Clafer-Definitionen**	**# unbeschränkt**	**# falsche untere Grenze**	**# falsche obere Grenze**	**# fälschl. unbeschränkt**	**# falsche untere Grenze**	**# falsche obere Grenze**	**# fälschl. unbeschränkt**	**# nach unten beschränkt**	**# nach oben beschränkt**	**# unbeschränkt**
BC	8	0	0	14	12	29	50	50	25	25	123
BMM	0	0	0	0	0	0	0	0	0	0	0
EDM	4	0	1	9	9	4	9	9	0	0	2
PerR	0	0	0	3	3	3	6	6	0	0	3
SAS	3	10	5	3	2	5	8	8	1	1	1
TM	6	0	0	8	0	4	11	11	2	2	2
TnT	7	8	0	11	6	5	3	3	0	0	5
Gesamt	28	18	6	48	32	50	87	87	28	28	136

Tabelle 7.5: Ergebnisse der Untersuchung der Anwendbarkeit unseres Analyseverfahrens

Weiterhin zeigen die Ergebnisse eine große Anzahl von Wertebereichen von Attributen, die für die erweiterte Kernsprache unbeschränkt sind. Für die Unbeschränktheit der Wertebereiche können zwei Ursachen unterschieden werden:

(i) Durch die Transformation in die erweiterte Kernsprache werden Constraints, die den Wertebereich des Attributs potentiell einschränken, entfernt. Das Attribut ist somit ausschließlich für die erweiterte Kernsprache unbeschränkt.

(ii) Das Attribut ist durch keinen Constraint gebunden und findet innerhalb der Spezifikation somit keine weitere Verwendung. Es ist daher ein Löschkandidat bei der Überarbeitung der Spezifikation.

Unser Analyseverfahren liefert somit Hinweise auf potentiell ungebundene Attribute innerhalb einer Constraint.

Darüber hinaus zeigen die Ergebnisse zahlreiche tote Clafer-Definitionen. Diese haben in den meisten Fällen folgende Ursachen:

(i) Falls eine konkrete Clafer-Definition eine abstrakte Clafer-Definition verfeinert, erbt sie neben den Constraint auch geschachtelte Sub-Clafer-Definitionen. In vielen

Fällen werden die geerbten Sub-Clafer-Definitionen aufgrund zusätzlicher Constraints niemals instanziiert und sind daher tote Clafer-Definitionen. Dies ist häufig gewünscht, kann jedoch auch ein Hinweis auf einen Spezifikationsfehler oder den Bedarf eines Refactorings der Spezifikation sein. So konnte zum Beispiel beim Inspizieren der Ursache einer identifizierten toten Clafer-Definition im System EDM ein Spezifikationsfehler durch die (vermutlich unerwünschte) Verfeinerung einer abstrakten Clafer-Definition festgestellt werden.

(ii) Eine tote Clafer-Definition kann aufgrund von Constraints durch Seiteneffekte anderer Clafer-Definitionen auftreten. Für die Spezifikation des in dieser Arbeit betrachteten selbst-adaptiven Kommunikationssystems tritt beispielsweise eine tote Clafer-Definition in der in Listing 3.1 gezeigten Spezifikation für die Clafer-Definition `Normal` auf. Diese Anomalie wird verursacht durch Wechselwirkungen von Multiplizitätsintervallen anderer Clafer-Definitionen und Constraints.

Zusätzlich liefert unser Analyseverfahren Informationen über die Unbeschränktheit von Clafer-Definitionen. Diese Informationen können verwendet werden, um den Suchraum für existierende Verfahren zur Instanzgenerierung automatisiert festzulegen.

7.5 UNTERSUCHUNG DER MODELLGRÖSSEN

In diesem Abschnitt untersuchen wir die Effizienz unseres Analyseverfahrens in Bezug auf die Größe der Spezifikationen und Zwischenrepräsentationen. Darüber hinaus untersuchen wir die Größe des resultierenden mathematischen Optimierungsproblems.

7.5.1 *Versuchsaufbau*

Wir untersuchen die Größen der Spezifikationen und Zwischenrepräsentation für die in Tabelle 7.4 gelisteten Systeme. Für jedes System betrachten wir die Größe der Spezifikation hinsichtlich der Anzahl der konkreten und abstrakten Clafer-Definitionen sowie der Anzahl der Constraints. Wir vergleichen die Größe der ursprünglichen Spezifikation mit der Größe der auf die Kernsprache reduzierten Spezifikation und der Größe der Spezifikation nach Abflachung der Vererbungshierarchie. Zur Beurteilung der Größe der Multimengen-Repräsentation betrachten wir die Anzahl der Multimengen und der Bedingungen zwischen Zählfunktionen, die durch die in Kapitel 5 gezeigte Transformationen erzeugt wurden. Zur Beurteilung der Größe des mathematischen Optimierungsproblems betrachten wir die Anzahl der Entscheidungsvariablen und (MILP-)Constraints.

7.5.2 *Ergebnisse*

Die Ergebnisse der Versuchsdurchführung sind in Tabelle 7.6 enthalten. Nach Reduktion einer vollständigen Clafer-Spezifikation in die erweiterte Kernsprache werden über alle

betrachteten Spezifikationen hinweg 28 Constraints verworfen. Im Falle vom System BC werden bis zu 25 Constraints verworfen. Die verbliebenen untersuchten Spezifikationen enthalten überwiegend Sprachelemente, die Teil der erweiterten Kernsprache sind. Insgesamt sind weniger als 10 % der verwendeten Constraints durch unser Analyseverfahren nicht analysierbar. Wie erwartet, ändert sich für alle untersuchten Systeme die Anzahl der Clafer-Definitionen nach Reduktion in die erweiterte Kernsprache nicht.

Die Abflachung der Vererbungshierarchie führt in allen Fällen - außer für das System BMM, das ausschließlich aus abstrakten Clafer-Definitionen besteht - zu einer Erhöhung der Anzahl der Clafer-Definitionen. Durch die Abflachung der Vererbungshierarchie erhöht sich die Anzahl der Clafer-Definitionen gegenüber der Spezifikation, die in der erweiterten Kernsprache vorliegt, im Mittel um ungefähr 11 %. Die Anzahl der Constraints erhöht sich durch die Abflachung der Vererbungshierarchie im Mittel um 40 %. Eine Ausnahme stellt das System BMM dar, das nur aus abstrakten Clafer-Definitionen besteht. Hier reduziert sich die Anzahl der Clafer-Definitionen durch die Abflachung der Vererbungshierarchie drastisch. Überdies werden in diesem Fall durch die Abflachung der Vererbungshierarchie sämtliche Constraints entfernt.

	Original		**Kernsprache**		**Abgeflacht**		**MILP**	
System	**# Clafer-Definitionen**	**# Constraints**	**# Clafer-Definitionen**	**# Constraints**	**# Clafer-Definitionen**	**# Constraints**	**# Variablen**	**# Constraints**
BC	185	225	185	200	314	242	9918	22146
BMM	156	19	156	18	4	0	9	6
EDM	41	14	41	14	55	30	510	722
PerR	13	12	13	12	28	34	584	1073
SAS	34	27	34	26	57	38	1468	3309
TM	34	10	34	10	34	10	180	314
TnT	22	12	22	11	46	53	1024	1734
Gesamt	485	319	485	291	538	407	13693	29304

Tabelle 7.6: Ergebnisse der Untersuchung des Einflusses unseres Analyseverfahrens auf Modellgrößen

Abbildung 7.2 zeigt den Zusammenhang der Anzahl der Clafer-Definitionen und der Anzahl der Constraints der abgeflachten Clafer-Spezifikationen zu der Anzahl der

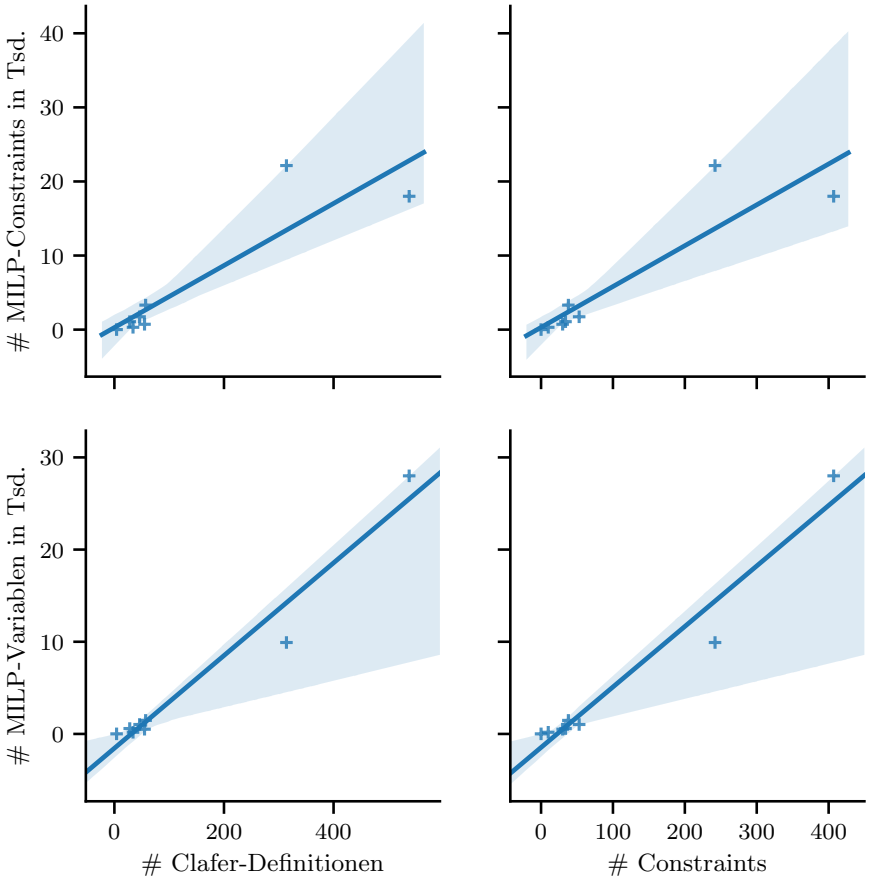


Abbildung 7.2: Ergebnisse der Evaluation des Einflusses der Spezifikationsgrößen auf die Größe des mathematischen Optimierungsproblems

Entscheidungsvariablen sowie der Anzahl der MILP-Constraints des resultierenden mathematischen Optimierungsproblems. Die mathematischen Optimierungsprobleme werden hinsichtlich der Anzahl der Entscheidungsvariablen und MILP-Constraints mit zunehmender Anzahl von Clafer-Definitionen und Constraints für alle untersuchten Systeme ungefähr in der gleichen Größenordnung vergrößert.

7.5.3 *Diskussion*

Die Forschungsfrage FF B.1 zur Untersuchung der Effizienz unseres Analyseverfahrens in Bezug auf Modellgrößen nach Transformationsschritten beantworten wir wie folgt: Die Ergebnisse zeigen, dass die Größe der Spezifikationen durch die Abflachung der Vererbungshierarchie deutlich erhöht wird. Im Fall des Systems BMM zeigen die Ergebnisse jedoch, dass die Abflachung der Vererbungshierarchie potentiell auch dazu geeignet

ist, die Größe von Spezifikationen zu reduzieren, indem abstrakte Clafer-Definitionen beseitigt werden, die niemals instanziiert werden.

Für die resultierenden mathematischen Optimierungsprobleme können wir eine Zunahme der Modellgrößen für alle betrachteten Systeme in einer ungefähr gleichen Größenordnung beobachten. Insbesondere bei Spezifikationen, die sehr viele Constraints mit langen Pfadausdrücken enthalten (wie im Falle des Systems BC), können wir eine deutliche Vergrößerung der resultierenden mathematischen Optimierungsprobleme beobachten. Um extrem große Spezifikationen mit sehr tiefen Pfadausdrücken handhabbar zu halten, besteht die Möglichkeit, Abschätzungen für sehr tiefe Pfadausdrücke nicht vorzunehmen, mit der Einschränkung, weniger Anomalien identifizieren zu können.

7.6 UNTERSUCHUNG DES RECHENAUFWANDS

In diesem Abschnitt untersuchen wir die Effizienz unseres Analyseverfahrens in Bezug auf den Rechenaufwand (FF B.2) im Vergleich zu existierenden Ansätzen Konsistenzprüfung (Instanzgenerierung) von Clafer-Spezifikationen.

7.6.1 *Versuchsaufbau*

Um die Forschungsfrage zu beantworten, vergleichen wir die Laufzeit der Konsistenzprüfung unseres Analyseverfahrens mit existierenden Ansätzen für repräsentative Clafer-Spezifikationen. Wie in Abschnitt 6.2.3 beschrieben, wird zur Konsistenzprüfung die minimale untere Schranke der Clafer-Definition c_r berechnet. Falls der Solver eine optimale Belegung der Entscheidungsvariablen findet, existiert mindestens eine Instanz der Clafer-Spezifikation. Bei existierenden Verfahren zur Generierung von Instanzen ist eine Clafer-Spezifikation genau dann konsistent, wenn eine Instanz gefunden werden kann. Als Baseline für die Messung des Rechenaufwandes verwenden wir die Alloy-basierte und Constraint-Programming-basierte Implementierung der Instanzgeneratoren (siehe Abschnitt 7.3). Für unser Analyseverfahren verwenden wir den MILP-Solver GUROBI [63].

Wir wiederholen jedes Experiment fünfmal und berechnen hieraus den Mittelwert. Als Zeitbeschränkung für einen Solver-Lauf verwenden wir 30 Minuten. Um die Vergleichbarkeit der Laufzeiten zu gewährleisten, nutzen wir als Eingabe der Baseline Clafer-Spezifikationen, die auf die erweiterte Kernsprache reduziert sind. Auf diese Weise werden nur Sprachbestandteile berücksichtigt, die Teil der erweiterten Kernsprache sind. Die gemessene Zeit eines Experiments entspricht der Summe der Zeiten, die für die Vorverarbeitung, Kompilierung und Lösung verwendet werden. Wir messen die Laufzeiten für jedes System in Tabelle 7.4. Zusätzlich, um die Laufzeiteffizienz der Analyse von Spezifikationen mit reellwertigen Attributen zu untersuchen, erzeugen wir ausgehend von den betrachteten Systemen weitere Spezifikationen, indem wir alle ganzzahligen durch reellwertige Attribute ersetzen. Die Experimente wurden auf einer Unix-Maschine mit Intel Core i5 (2.9 GHz, 8 GB RAM) ausgeführt.

7.6.2 *Ergebnisse*

Die Ergebnisse der Versuchsdurchführung für die betrachtete Menge von Clafer-Spezifikationen ist in Tabelle 7.7 zu sehen. Unser Analyseverfahren benötigt für einen Analyselauf zwischen 146 ms (BMM) und 16.263 ms (BC) an CPU-Zeit. Im Vergleich dazu benötigt der Constraint-Programming-basierte Ansatz CHOCOSOLVER zur Instanzgenerierung zwischen 401 ms (BMM) und 4.074 ms (BC). Das Alloy-basierte Analyseverfahren benötigt zwischen 138 ms (TM) und 181.645 ms (BC). Deutliche Differenzen zwischen der Laufzeit der Analyseverfahren sind insbesondere für das System BC feststellbar: Hier benötigt das Analyseverfahren CHOCOSOLVER im Vergleich zu unserem Analyseverfahren nur ein Viertel der CPU-Zeit. Unser Analyseverfahren benötigt 16.263 ms, wobei hiervon im Mittel ca. 12.000 ms für die Vorverarbeitung und Kompilierung aufgewendet werden. Die Netto-Solver-Laufzeit unseres Analyseverfahrens entspricht ungefähr der Laufzeit des Analyseverfahrens CHOCOSOLVER.

Für die Spezifikation des in dieser Arbeit betrachteten selbst-adaptiven Kommunikationssystems (SAS) konnten aufgrund mangelnder Unterstützung reellwertiger Attribute weder mit ALLOY noch CHOCOSOLVER Konsistenzprüfungen durchgeführt werden. Unser Analyseverfahren benötigt im Mittel 1.468 ms für die Konsistenzprüfung der in Listing 3.1 gezeigten Spezifikation.

System	Alloy [ms]	Chocosolver [ms]	Boundanalyzer [ms]
BC	181.645	4.074	16.263
BMM	1.101	2.076	618
EDM	1.693	1.115	212
PerR	1.646	401	146
SAS	*	*	1.468
TM	138	616	298
TnT	4.706	581	388

Tabelle 7.7: Ergebnisse der Untersuchung der Laufzeiteffizienz unseres Analyseverfahrens

Abbildung 7.3 zeigt die Ergebnisse der Evaluation der Laufzeit unseres Analyseverfahrens für Spezifikationen, die ganzzahlige und reellwertige Attribute enthalten. Zu sehen ist zudem der Zusammenhang zwischen Solver-Laufzeit sowie der Gesamtlaufzeit und

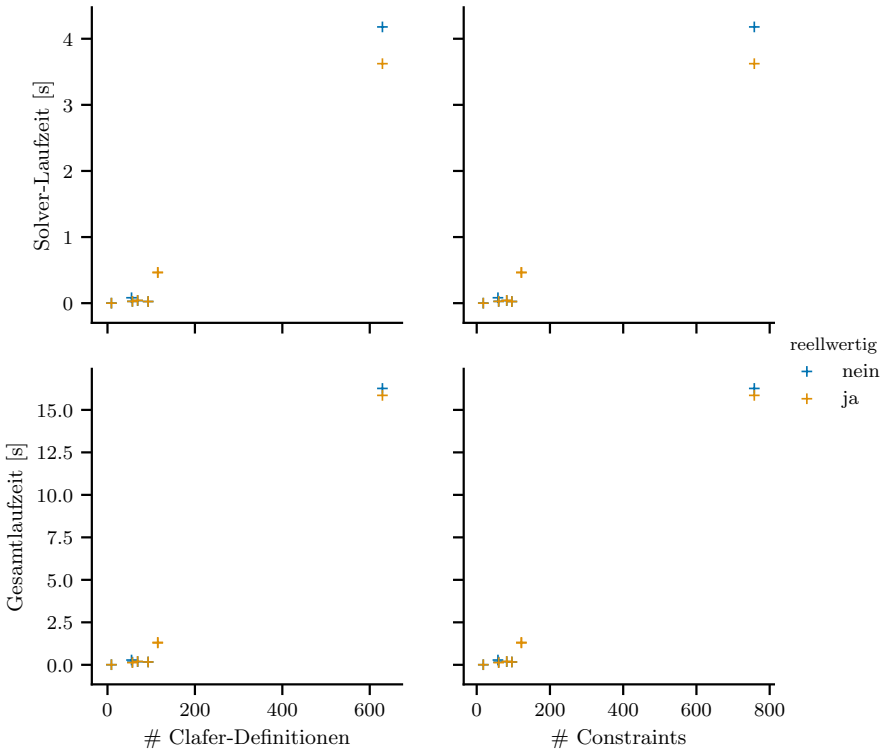


Abbildung 7.3: Ergebnisse der Evaluation des Rechenaufwandes unseres Analyseverfahrens

der Anzahl der Clafer-Definitionen sowie der Anzahl der Constraints (der abgeflachten Clafer-Spezifikationen). Die Ergebnisse zeigen, dass durch die Ersetzung von ganzzahligen durch reellwertige Attribute keine Laufzeitverschlechterungen eintreten. Zudem zeigt sich (erwartungsgemäß) ein deutlicher Zusammenhang zwischen Problemgröße und Laufzeit unseres Analyseverfahrens.

7.6.3 *Diskussion*

Die Forschungsfrage FF B.2 zur Untersuchung der Laufzeiteffizienz unseres Analyseverfahrens beantworten wir wie folgt: Unser Analyseverfahren zeigt, insbesondere mit zunehmender Größe der Spezifikation, deutliche Laufzeitverbesserungen im Vergleich zum Alloy-basierten Analyseverfahren. Für kleine Modellgrößen zeigt unser Analyseverfahren eine vergleichbare oder teilweise geringere Laufzeit gegenüber dem Constraint-Programming basierten Analyseverfahren. Für das betrachtete System BC benötigt die Kompilierung der Spezifikation und Erzeugung des mathematischen Optimierungsproblems einen Großteil der Laufzeit (12.000 *ms* Gesamtlaufzeit gegenüber 4.000 ms Solver-Laufzeit). Dies macht sich insbesondere bemerkbar bei einmaligen Solver-Auf-

rufen, die keine Möglichkeiten der Anwendung von Caching-Strategien ermöglichen. Für die Analyse aller Multiplizitäts- und Gruppenkardinalitätsintervalle sowie Attribute muss das mathematische Optimierungsproblem einmalig generiert werden und wird ansonsten zwischen den Solver-Aufrufen wiederverwendet. Wir nehmen an, dass dieses Anwendungsszenario der Regelfall bei der Validierung von Clafer-Spezifikationen gegenüber einzelnen Solver-Aufrufen darstellt. Bei großen Modellen sind die Laufzeiten von ChocoSolver und unseres Analyseverfahrens somit vergleichbar, wenn der einmalige Vorverarbeitungsschritt nicht berücksichtigt wird.

Die Evaluation hat gezeigt, dass die Repräsentation der Analyseziele als mathematisches Optimierungsproblem geeignet ist, Spezifikationen mit reellwertigen Attributen zu analysieren. Unser Analyseverfahren zeigt insbesondere Effizienzverbesserungen gegenüber existierenden Verfahren, die Analyseziele als SAT- oder Constraint-Programming-Probleme repräsentieren und bei Clafer-Spezifikationen, die Attribute mit großen ganzzahligen Wertebereichen aufweisen. Im Gegensatz zu existierenden Verfahren ist unser Analyseverfahren zudem in der Lage, Spezifikationen mit reellwertigen Attributen zu analysieren. Bei vergleichbaren Laufzeiten liefert unser Analyseverfahren garantiert korrekte Angaben zu Anomalien, die bei ChocoSolver und Alloy eine richtige Wahl des Suchraums voraussetzen. Unser Analyseverfahren ist zudem in der Lage Anomalien zu identifizieren, die von den genannten existierenden Ansätzen nicht betrachtet werden.

7.7 GEFÄHRDUNG DER VALIDITÄT DER ERGEBNISSE

GEFÄHRDUNG DER INTERNEN VALIDITÄT: Eine Gefährdung der internen Validität der Ergebnisse könnte sich aus der Auswahl der analysierten Anomalietypen ergeben. Da Anomaliedetektion in Clafer-Spezifikationen ein neues Forschungsfeld ist und uns nicht bekannt ist, dass zuvor entsprechende Anomalietypen für Clafer-Definitionen definiert wurden, greifen wir zur Ableitung der in dieser Arbeit vorgestellten Anomalien auf etablierte Anomalietypen zurück, die bei der Validierung von kardinalitätsbasierten Feature-Modellen im Problemraum und UML-Klassendiagrammen im Lösungsraum verbreitet sind. Eine weitere Gefährdung der internen Validität könnte sich aus der Verzerrung der Ergebnisse der Messungen der Laufzeiten ergeben. Um Verzerrungen bei der Messung der Laufzeiten zu verringern, haben wir die Experimente fünfmal wiederholt und den Mittelwert aus allen gemessenen Laufzeiten berechnet.

GEFÄHRDUNG DER EXTERNEN VALIDITÄT: Eine Gefährdung der externen Validität der Ergebnisse könnte sich aus dem fehlenden Vergleich mit anderen Ansätzen zur Anomaliedetektion in Clafer-Spezifikationen ergeben. Die Gefährdung ergibt sich aus der Neuheit unseres Ansatzes und fehlenden Vergleichsmöglichkeiten. Uns ist kein vergleichbarer Ansatz bekannt, der in der Lage ist, Anomalien in Clafer-Spezifikation zu identifizieren. Um die Gefährdung der externen Validität abzuschwächen, greifen wir auf existierende Verfahren zur Konsistenzprüfung zurück und generieren automatisch

entsprechend präparierte Spezifikationen, um die Existenz identifizierter Anomalien zu verifizieren.

Eine weitere Gefährdung der externen Validität der Ergebnisse könnte sich aus der Auswahl der untersuchten Systeme ergeben. Bei den ausgewählten Systemen handelt es sich zum Großteil um publizierte Fallstudien, die unserer Meinung nach eine angemessene Größe und Komplexität aufweisen. Weiterhin könnte aufgrund der geringen Anzahl der betrachteten Systeme eine externe Gefährdung der Validität vorliegen. Die Ergebnisse zeigen die Effektivität und Effizienz bereits für eine Reihe von Systemen mit variierender Größe und Komplexität aus unterschiedlichen Domänen. Für eine umfassende Verallgemeinerung der Ergebnisse muss unser Analyseverfahren jedoch auf weitere Systeme angewendet werden, die allerdings zum Zeitpunkt des Verfassens dieser Arbeit nicht vorlagen.

8

DISKUSSION VERWANDTER ARBEITEN

In diesem Kapitel diskutieren wir verwandte Arbeiten zu dem in dieser Arbeit vorgestellten Analyseverfahren. Zunächst diskutieren wir verwandte Arbeiten zur Analyse von Problemraumspezifikationen. In einem zweiten Schritt diskutieren wir verwandte Arbeiten zur Analyse struktureller Lösungsraumspezifikationen. Schließlich diskutieren wir verwandte Arbeiten zur Analyse integrierter Software-Produktlinien-Spezifikationen.

8.1 ANALYSE VON PROBLEMRAUMSPEZIFIKATIONEN

Die verwendete Analysetechnik für die Identifikation von Inkonsistenzen und Anomalien ist stark von der Ausdrucksstärke der zugrundeliegenden Spezifikationssprache abhängig. Im Folgenden diskutieren wir daher zunächst verwandte Arbeiten zur Analyse von Feature-Modellen mit binären Features. Anschließend diskutieren wir verwandte Arbeiten zur Analyse von attributierten und kardinalitätsbasierten Feature-Modellen, die in Abschnitt 2.2.4 unter dem Begriff erweiterter Feature-Modelle eingeführt wurden.

Analyse von Feature-Modellen mit binären Feature-Typen

Für die Analyse von Feature-Modellen mit binären Features lassen sich Ansätze unterscheiden, welche die Konfigurationssemantik entweder algebraisch, als Erfüllbarkeitsproblem oder als Constraint-Erfüllbarkeitsproblem formulieren. Schobbens et al. [135] und Heymans et al. [69] schlagen eine algebraische Charakterisierung der Konfigurationssemantik vor. Dies ermöglicht zwar die formale Charakterisierung von Analysezielen und Inkonsistenzen, stellt jedoch im Gegensatz zu unserem Analyseverfahren keinen konstruktiven Ansatz zur Identifikation von Inkonsistenzen und Anomalien zur Verfügung. Einige Arbeiten [16, 102] schlagen die Codierung der Konfigurationssemantik von Feature-Modellen mit ausschließlich binären Feature-Typen als Erfüllbarkeitsproblem bzw. Constraint-Erfüllbarkeitsproblem [19] vor. Wie in Abschnitt 2.2.5 beschrieben, werden Features hierbei als binäre Variablen codiert und Syntaxelemente der Feature-Diagramme in aussagenlogische Verknüpfungen transformiert. Die genannten Ansätze eignen sich, um Feature-Modelle mit ausschließlich binären Features zu analysieren.

Unser Analyseverfahren kann ebenso für die Analyse von Feature-Modellen mit ausschließlich binären Feature-Typen verwendet werden. Moderne Mixed Integer Linear Programming (MILP)-Solver erkennen das zugrunde liegende Optimierungsproblem mit ausschließlich binären Variablen und wenden Lösungsverfahren an, die zu SAT-basierten Lösungsverfahren vergleichbar effizient sind. Im Gegensatz zu den genannten Arbeiten unterstützt unser Analyseverfahren darüber hinaus die Analyse von weitaus ausdrucksstärkeren Spezifikationen, die in der Sprache Clafer formuliert sind.

Zahlreiche Arbeiten präsentieren Verfahren zur Identifikation von Anomalien vom Typ *totes Feature* und *fälschlicherweise-optionalen Features* in Feature-Modellen mit ausschließlich binären Features [20]. Batory et al. [16], Trinidad et al. [141] und Kramer et al. [87] schlagen darüber hinaus Verfahren zur Erklärung von Anomalieursachen vor. Felfernig et al. [54] liefern zwar Erklärungen für Anomalieursachen, beziehen diese aber nicht auf die Struktur des Feature-Modells. Kowal et al. [86] schlagen ein Verfahren vor, das darüber hinaus redundante Constraints erkennt und Faktoren für die Ursachen von Anomalien priorisiert. Nieke et al. [106] berücksichtigen zusätzlich Änderungshistorien von Feature-Modellen und untersuchen Anomalien, die sich aus Änderungen eines Feature-Modells ergeben. Im Gegensatz zu den genannten Arbeiten stellt unser Analyseverfahren zwar Informationen dazu bereit, welche Elemente von Anomalien betroffen sind, jedoch liefert es keine Erklärung, welche Constraints oder Spezifikationselemente für das Entstehen der Anomalie ursächlich sind. Darüber hinaus liefert unser Analyseverfahren zwar Informationen über Constraint-Verletzungen, redundante Constraints, die den Suchraum nicht zusätzlich einschränken, werden jedoch nicht automatisiert erkannt. Beide genannten Punkte stellen mögliche Erweiterungen unseres Analyseverfahrens dar.

Analyse kardinalitätsbasierter Feature-Modelle

Riebisch et al. [125] schlagen erstmals vor, herkömmliche Feature-Modelle mit binären Feature-Typen um Multiplizitäten zu erweitern. Czarnecki et al. erweitern Feature-Modelle um Gruppenkardinalitätsintervalle, lassen jedoch keine Kombination von Feature-Instanz-Kardinalitätsintervallen und Gruppeninstanz-Kardinalitätsintervallen zu, welche die Konfigurationssemantik im Vergleich zur in dieser Arbeit präsentierten Semantik der Sprache Clafer wesentlich vereinfacht [44]. Auf den zuvor genannten Arbeiten basierend definieren Czarnecki et al. [45] eine erweiterte Konfigurationssemantik, die auf dem Klonen von Teilbäumen basiert, und schlagen eine Übersetzung in eine kontextfreie Grammatik vor. Hierbei können erstmals auch unbeschränkte Kardinalitätsintervalle spezifiziert werden. Implikationen daraus auf die Konfigurationssemantik werden jedoch nicht beachtet.

Quinton et al. [119] führen Quell- und Ziel-Kardinalitätsintervalle für Require-Constraints ein. Im Gegensatz zur in dieser Arbeit verwendeten Sprache Clafer, können jedoch keine ausdrucksstärkeren Constraints spezifiziert werden. Darüber hinaus können Feature-Instanz- und Gruppeninstanz-Kardinalitäten nicht kombiniert werden. Quinton

et al. erwähnen zwar unbeschränkte Multiplizitätsintervalle, diese werden aber weder durch die Konfigurationssemantik noch als Teil des Analyseverfahrens berücksichtigt.

Michel et al. [105] untersuchen semantische Mehrdeutigkeiten, die durch die Kombination von Feature-Instanz- und Gruppentyp-Kardinalitätsintervallen entstehen können. Sie unterscheiden darüber hinaus erstmals eine lokale Feature-Instanz-basierte und globale Feature-Typ-basierte Interpretation von Multiplizitätsintervallen. Sie schlagen jedoch die ausschließliche Verwendung der globalen Interpretation vor. Im Gegensatz hierzu, erlaubt unser Analyseverfahren sowohl die Analyse lokaler kontextabhängiger als auch globaler kontextunabhängiger Constraints. Cordy et al. [41] erlauben die Kombination von Feature- und Gruppentyp-Kardinalitätsintervalle, berücksichtigen jedoch keine Gruppeninstanz-Kardinalitätsintervalle. Im Gegensatz hierzu erlaubt die in dieser Arbeit verwendete Sprache Clafer nur Gruppenkardinalitäts- und Multiplizitätsintervallen zu spezifizieren, welche in kardinalitätsbasierten Feature-Modellen Gruppeninstanz- und Feature-Instanz-Kardinalitätsintervallen entsprechen. Gruppentyp-Kardinalitätsintervalle werden syntaktisch zwar nicht unterstützt, können aber als zusätzliche Constraints codiert werden. Keine der genannten Arbeiten betrachtet die Unbeschränktheit von Multiplizitätsintervallen in der Konfigurationssemantik oder als Analyseziel.

Quinton et al. [116, 117, 119] schlagen ein Analyseverfahren zur Erkennung von Anomalien in kardinalitätsbasierten Feature-Modellen vor. Der hierbei vorgeschlagene Anomalietyp ähnelt der in dieser Arbeit verwendeten Charakterisierung von ungültigen Wertebereichen in Multiplizitätsintervallen. Darüber hinaus betrachten Quinton et al. im Gegensatz zu dieser Arbeit auch Inkonsistenzen, die durch die Evolution einer Spezifikation entstehen können. Für die Analysen wird das zugrundeliegende Analyseproblem als Constraint-Erfüllbarkeitsproblem formuliert, das mithilfe von Standard-Lösungsverfahren gelöst werden kann, jedoch im Gegensatz zu unserem Analyseverfahren keine unbeschränkten Multiplizitätsintervalle zulässt. Cordy et al. [41] und Zhang et al. [159] präsentieren Verfahren zur Konsistenzprüfung von kardinalitätsbasierten Feature-Modellen. Hierbei wird das zugrundeliegende Analyseproblem als Binary Decision Diagram formuliert. Keines der genannten Verfahren erlaubt jedoch die Analyse von Spezifikationen mit unbeschränkten Multiplizitätsintervallen.

Analyse attributierter Feature-Modelle

Wie in Abschnitt 2.2.4 beschrieben wurde, erweitern attributierte Feature-Modelle die Konfigurationssemantik herkömmlicher Feature-Modelle um numerische Features, die entweder ganzzahlige oder reellwertige Domänen aufweisen und zusätzlich auf einen Wertebereich eingeschränkt sein können [110]. Zahlreiche verwandte Arbeiten existieren zur Validierung und Analyse attributierter Feature-Modelle.

Lesta et al. [92] schlagen ein Verfahren zur Erkennung von Anomalien vom Typ *totes Feature* und *falsch-optionales Feature* in attributierten Feature-Modellen vor. Darüber hinaus identifiziert ihr Verfahren widersprüchliche Constraints, die Auslöser einer Anomalie

sind, und somit als Erklärung der Anomalieursache dienen. Das zugrundeliegende Analyseproblem wird als Constraint-Erfüllbarkeitsproblem codiert, das mithilfe von Standard-Lösungsverfahren gelöst werden kann. Im Gegensatz zu unserem Analyseverfahren werden jedoch nur ganzzahlige numerische Features unterstützt, deren Wertebereich zudem beschränkt sein muss.

Zaid et al. [157] schlagen einen Ansatz zur Konsistenzprüfung attributierter Feature-Modelle vor. Das zugrundeliegende Analyseproblem wird in Beschreibungslogik formuliert. Im Gegensatz zu unserem Analyseverfahren werden jedoch nur ganzzahlige Attribute unterstützt, deren Wertebereich beschränkt ist. Zusätzlich können nur Vergleichsoperationen zwischen numerischen Features formuliert werden. Im Gegensatz zu unserem Analyseverfahren ist die Ausdrucksstärke der Constraints somit stark limitiert.

Karatas et al. [80, 81] schlagen einen Ansatz zur Übersetzung erweiterter Feature-Modelle in Constraint-Erfüllbarkeitsprobleme vor. Ähnlich zu unserem Analyseverfahren werden hierbei komplexe arithmetische Constraints unterstützt, die Einschränkungen über Feature-Typen und Attributen formulieren. Im Gegensatz zu unserem Analyseverfahren werden auch nicht-lineare Modulo- oder Divisionsoperationen unterstützt. In dem von Karatas et al. vorgeschlagenen Ansatz werden Constraints jedoch nur global interpretiert, kontextabhängige Einschränkungen können dementsprechend nicht charakterisiert werden. Weiterhin können keine Gruppeninstanz-Kardinalitätsintervalle spezifiziert werden. Eine Kombination von Feature-Instanz- und Gruppentyp-Kardinalitätsintervallen ist darüber hinaus nicht zulässig. Im Gegensatz zu unserem Analyseverfahren werden nur numerische Features mit ganzzahliger Domäne unterstützt, die zudem einen beschränkten Wertebereich aufweisen müssen. Die Spezifikation unbeschränkter Multiplizitätsintervalle multiinstanziierbarer Features wird zwar nicht explizit ausgeschlossen, aber weder semantisch noch durch die Analyse adressiert.

Weitere Ansätze unterstützen zusätzlich die Spezifikation dynamischer Bindungszeiten und Einschränkungen zwischen Bindungszeiten aus DSPLs. Mennicke et al. [103] erlauben die Annotation von Feature-Typen mit Bindungszeiten und schlagen eine Repräsentation der Rekonfigurationssemantik als Workflow-Petri-Netz vor, um mehrstufige Konfigurationsprozesse zu berechnen. Hierbei werden jedoch nur binäre Features betrachtet. Bürdek et al. [29] erlauben ebenfalls die Annotation von Feature-Typen mit Bindungszeiten und ermöglichen zudem die Spezifikation ganzzahliger numerischer Features mit beschränkten Wertebereichen. Zur Validierung und Analyse schlagen Bürdek et al. vor, Wertebereiche von Attributen in Alternativ-Gruppen mit binären Features zu übersetzten. Constraints werden als Cross-Tree-Constraints zwischen binären Features codiert. Lochau et al. [94] schlagen darauf basierend eine Repräsentation der Rekonfigurationssemantik als Zustandsautomaten vor, um potentiell unendliches Rekonfigurationsverhalten mithilfe von Model-Checking-Technologien auf Lebendigkeits- und Fortschrittseigenschaften zu überprüfen. Die genannten Ansätze unterstützen im Gegensatz zu unserem Analyseverfahren nur numerische Features mit begrenzten Wertebereichen. Weiterhin ermöglicht unser Analyseverfahren die Validierung und Analyse

struktureller Abhängigkeiten: Abhängigkeiten zwischen Bindungszeitpunkten können durch unser Verfahren nicht analysiert werden. Die Berücksichtigung von Abhängigkeiten zwischen Bindungszeitpunkten stellt daher eine mögliche Erweiterung unseres Analyseverfahrens dar.

8.2 ANALYSE VON LÖSUNGSRAUMSPEZIFIKATIONEN

Für die Spezifikation der Konsistenzeigenschaften der Lösungsraumarchitektur wurden in dieser Arbeit UML-Klassendiagramme und Ausdrücke in OCL verwendet. Im Folgenden liegt daher der Fokus auf der Diskussion verwandter Arbeiten, in denen Analysetechniken für UML-Klassendiagramme und OCL-Ausdrücke präsentiert werden. Zusätzlich diskutieren wir verwandte Arbeiten zur Analyse von Entity-Relationship (ER)-Diagrammen, die sich zur Spezifikation von Datenbank-Schemata etabliert haben.

Analyse von UML-Klassendiagrammen

Bei der Analyse von UML-Klassendiagrammen können Arbeiten anhand der semantischen Repräsentation und der unterstützten Sprachelemente unterschieden werden.

Cadoli et al. [34] schlagen einen Ansatz zur Konsistenzprüfung von UML-Klassendiagrammen vor, der das zugrundeliegende Analyseproblem mithilfe von Constraint-Programming-Techniken als Constraint-Erfüllbarkeitsproblem formuliert, das mit Hilfe von Standard-Lösungsverfahren gelöst werden kann. Als Analyseproblem wird von Cadoli et al. betrachtet, ob eine Spezifikationsinstanz existiert, in der eine Klasse entweder keine oder unendlich viele Objekte repräsentieren muss. In diesem Fall weist die Klasse eine Inkonsistenz auf. Ähnlich zu unserem Analyseverfahren wird der Suchraum durch eine Menge linearer Ungleichungen begrenzt, sodass keine expliziten Grenzen des Suchraums angegeben werden müssen. Im Gegensatz zu unserem Analyseverfahren werden jedoch nur Multiplizitätsintervalle von Assoziationsbeziehungen und Generalisierungsbeziehungen berücksichtigt.

Berardi et al. [24] und Cali et al. [35] verwenden Beschreibungslogik, um formale Analyseziele von UML-Klassendiagrammen zu charakterisieren, die Aussagen über die Konsistenz des Klassendiagramms und Konsistenz von Klassen sowie die Äquivalenz von Klassen zulassen. Im Gegensatz zu dem in dieser Arbeit gewählten Vorgehen, können keine Einschränkungen durch zusätzliche Constraints spezifiziert werden, die in Form einer (entscheidbaren) Teilmenge einer Constraint-Sprache (z. B. OCL für UML-Klassendiagramme) formuliert sind.

Eine Reihe von verwandten Arbeiten schlagen die Übersetzung von UML-Klassendiagrammen in die strukturelle Spezifikationssprache Alloy [76] vor. Anastasakis et al. [5] und Shah et al. [136] schlagen vor, Sprachbestandteile von UML-Klassendiagrammen äquivalenten Sprachbestandteilen von Alloy zuzuordnen, und ermöglichen so grundlegende Konsistenzprüfungen. Allerdings lassen sich durch das gewählte Vorgehen nur

Sprachbestandteile von UML-Klassendiagrammen übersetzen, die eine unmittelbare Entsprechung in Alloy haben. Es wird somit nur ein sehr limitierter Sprachumfang von UML-Klassendiagrammen unterstützt. Maoz et al. [98] schlagen demgegenüber die Übersetzung von UML-Klassendiagramme in eine Zwischenrepräsentation vor, die nicht von Alloy unterstützte Sprachbestandteile als zusätzliche Constraints und Relationen zwischen zusätzlichen Modellelementen formuliert. Im Gegensatz zu der Sprache Clafer, die als Eingabe für unser Analyseverfahren dient, können auf diese Weise auch erweiterte Sprachbestandteile von UML-Klassendiagrammen wie Mehrfachvererbung repräsentiert werden. Darüber hinaus lassen sich so Analyseziele wie die Berechnung von Differenzen zwischen UML-Klassendiagrammen formulieren. Dies wird durch unser Analyseverfahren nicht unterstützt.

Alle genannten Arbeiten, die UML-Klassendiagramme in Alloy-Repräsentationen übersetzen, haben gemeinsam, dass explizite Obergrenzen für die maximale Anzahl jedes Elements der Spezifikation angegeben werden müssen, um den Suchraum der Instanzsuche zu begrenzen. Dies führt dazu, dass – falls eine Inkonsistenz identifiziert wird – nicht entschieden werden kann, ob dies aufgrund eines zu klein gewählten Suchraums oder aufgrund einer tatsächlichen Inkonsistenz der Fall ist. Im Gegensatz hierzu erlaubt unser Verfahren eine vollständige Analyse für einen entscheidbaren Kern der Sprache Clafer. Darüber hinaus ist zu erwarten, dass die Repräsentation von Attributen oder sonstigen numerischen Ausdrücken in Alloy zu Effizienzproblemen führt, da Alloy als Zielrepräsentation der Analyse die Spezifikation als aussagenlogische Formel eines Erfüllbarkeitsproblems übersetzt. Im Gegensatz hierzu verwendet unser Analyseverfahren als Zielrepräsentation eine Formulierung als mathematisches Optimierungsproblem, das neben numerischen Abhängigkeiten, die als Ungleichungen repräsentiert werden, auch Ausdrücke über reellwertige Attribute effizient handhaben kann.

Balaban et al. [11, 12, 13] und Maraee et al. [99] präsentieren in ihren Arbeiten spezialisierte Algorithmen, um die Erfüllbarkeit von UML-Klassendiagrammen zu prüfen. Der Fokus liegt hierbei auf der Analyse der Einschränkungen auf Multiplizitätsintervallen von Assoziationsbeziehungen, die sich durch Generalisierungsbeziehungen und Kompositionsbeziehungen ergeben. Ähnlich zu unserem Analyseverfahren charakterisieren Balaban et al. redundante Teilintervalle von Multiplizitätsintervallen von Assoziationsbeziehungen, für die keine erfüllbare Spezifikationsinstanz existiert und die durch die vorgeschlagene Analyse eingeschränkt werden sollen. Im Gegensatz zu unserem Analyseverfahren werden nur strukturelle Einschränkungen betrachtet, die sich durch die genannten Beziehungstypen in UML-Klassendiagrammen ergeben.

Feinerer et al. [51, 52, 53] schlagen in ihren Arbeiten einen Ansatz zur Konsistenzprüfung von UML-Klassendiagrammen vor, bei dem die Semantik von UML-Klassendiagrammen als Menge von Ungleichungen über ganzzahlige Werte repräsentiert wird. Hierbei werden Einschränkungen durch Multiplizitätsintervalle von Assoziationsbeziehungen unterstützt. Zusätzlich werden die UML-Eigenschaften *unique* und *non-unique* berücksichtigt, die an Enden von Assoziationsbeziehungen annotiert sind und mehrwerti-

ge Assoziationsbeziehungen zwischen Objekten präzisieren [128]. Ähnlich zum Vorgehen in dieser Arbeit dient ein mathematisches Optimierungsproblem, das mittels Integer Linear Programming (ILP)-Lösungsverfahren gelöst werden kann, als Zielrepräsentation. Darüber hinaus schlagen Feinerer et al. auf Basis der ILP-Formulierung vor, Spezifikationsinstanzen zu berechnen, die eine minimale Anzahl von Objekten aufweisen. Dies ist in unserem Verfahren ebenfalls erreichbar, indem als Zielfunktion der in Abschnitt 6.2 vorgestellten MILP-Formulierung die Summe aller Clafer-Definitionen minimiert wird, die in einer Clafer-Spezifikation enthalten sind. Im Gegensatz zu unserem Verfahren ermöglichen Feinerer et al. jedoch nicht die Analyse von UML-Klassendiagrammen mit komplexen Constraints auf Anomalien und Inkonsistenzen.

Analyse von UML-Klassendiagrammen mit OCL-Ausdrücken

Gogolla et al. [60] präsentieren zur Analyse von UML-Klassendiagrammen mit OCL-Ausdrücken das Werkzeug USE, das UML-Klassendiagramme und OCL-Ausdrücke in Java-Programmcode übersetzt und zur Konsistenzprüfung ausführt. Ein ähnliches interpreterbasierteres Vorgehen schlagen Husmann et al. [73] vor. Das Werkzeug Dresden OCL compiler setzt dieses Vorgehen um. Der Suchraum der Instanzsuche wird bei beiden Ansätzen durch eine zeitliche Beschränkung limitiert. Beide Ansätze unterstützen einen Großteil des Sprachumfangs von UML-Klassendiagrammen und OCL. Im Gegensatz zum Vorgehen in dieser Arbeit liegt der Fokus der genannten Ansätze auf der Generierung zulässiger Spezifikationsinstanzen. Sie sind aber nicht geeignet, um die Anomalietypen, die in Abschnitt 3.2 präsentiert wurden, zu identifizieren.

Cabot et al. [31, 32] präsentieren das Werkzeug UMLtoCSP, das UML-Klassendiagramme mit OCL-Constraints in Constraint-Erfüllbarkeitsprobleme übersetzt und diese auf Konsistenz sowie auf Vorhandensein redundanter Constraints prüft. Entscheidbarkeit wird erzielt, indem endliche Grenzen für die Anzahl von Instanzen und Wertebereiche von Attributen festgelegt werden und somit der Suchraum für das Lösungsverfahren begrenzt wird. Das von Cabot et al. vorgeschlagene Analyseverfahren ist unvollständig, da, falls keine Spezifikationsinstanz gefunden wird, keine Aussage möglich ist, ob der Suchraum zu klein gewählt oder die Spezifikation inkonsistent ist. Die in dieser Arbeit betrachtete Sprache Clafer hat durch die Möglichkeit der Spezifikation von Constraints in Prädikatenlogik erster Stufe eine vergleichbare Ausdrucksmächtigkeit wie die für die Spezifikation von Konsistenzeigenschaften in UML-Klassendiagrammen verbreiteten Sprachkonstrukte von OCL. Im Gegensatz zu dem Ansatz von Cabot et al. [31, 32] werden bei unserem Vorgehen Clafer-Spezifikationen auf einen entscheidbaren Kern reduziert, für den unser Analyseverfahren vollständig und korrekt ist. Dies reduziert zwar die Menge analysierbarer Spezifikationselemente, ermöglicht jedoch, wie in Abschnitt 6.1.3 präsentiert, trotzdem die Validierung jeder Spezifikation auf Anomalien.

Yu et al. [156] schlagen ein Verfahren vor, um Größenabschätzungen für OCL-Spezifikationen vorzunehmen. Hierbei werden Collection-Typen in OCL auf ganzzahlige

Variablen abgebildet und Ausdrücke beibehalten, die sich auf die Größe einer Collection beziehen oder die Größe einer Collection einschränken. Die genannte Abstraktion weist somit Ähnlichkeiten zu der in dieser Arbeit verwendeten Multimengen-Repräsentation auf. Im Gegensatz zu unserem Analyseverfahren verwenden Yu et al. symbolische Model-Checking-Technologien als Zielrepräsentation, mithilfe derer nach Falsifizierungen von invarianten OCL-Ausdrücken gesucht wird. Dies ermöglicht zwar ähnlich wie in unserem Analyseverfahren Konsistenzprüfungen, aber keine Identifikation von Anomalietypen, wie sie in dieser Arbeit vorgestellt wurden.

Analyse von Entity-Relationship-Diagrammen

Eine Reihe von Arbeiten betrachten Verfahren zur Analyse von ER-Diagrammen, die sich insbesondere für Datenbanken zur Spezifikation von Domänenmodellen etabliert haben. Lenzerini et al. [91] übersetzen ER-Schemata in Form eines linearen Ungleichungssystems und zeigen, dass ein ER-Schema genau dann erfüllbar ist, wenn für das Gleichungssystem Lösungen existieren. Boufares et al. [26] betrachten Konsistenzeigenschaften in ER-Schemata von Datenbanksystemen und schlagen ebenfalls eine Formulierung als lineares Ungleichungssystem vor, das mithilfe des Fourier-Motzkin-Eliminierungsverfahrens gelöst werden kann. Die genannten Arbeiten verwenden ähnlich wie unser Vorgehen lineare Ungleichungssysteme als Zielrepräsentation. Im Gegensatz zu Clafer-Spezifikationen, die unser Verfahren analysiert, weisen ER-Diagramme jedoch eine geringere Komplexität auf. Zudem werden im Gegensatz zu unserem Analyseverfahren nur Konsistenzprüfungen durchgeführt.

8.3 ANALYSE VON CLAFER-SPEZIFIKATIONEN

Die beträchtliche Ausdrucksstärke der Sprache Clafer macht die automatisierte Analyse von Clafer-Spezifikation auf Konsistenz herausfordernd. Existierende Ansätze zur Konsistenzprüfung von Clafer-Spezifikationen charakterisieren die Semantik entweder in der strukturellen Spezifikationssprache Alloy [76], in Prädikatenlogik erster Stufe (SMT) [15] oder mittels Constraint-Programming-Techniken [6]. Für die genannten Ansätze müssen die Suchräume der Instanzsuche vorab begrenzt werden. Dies kann entweder manuell oder durch Anwendung von Heuristiken, wie beispielsweise Feasibility-Based Bound Tightening (FBBT) [132] erfolgen.

Alle genannten Ansätze sind im Gegensatz zu unserem Analyseverfahren inhärent unvollständig, da sie Ergebnisse liefern, für die nicht bekannt ist, ob der Suchraum groß genug gewählt wurde. Für unser Analyseverfahren charakterisieren wir stattdessen eine reduzierte Kernsprache, für die eine korrekte und vollständige Analyse möglich ist.

9

ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde ein Ansatz zur automatisierten Analyse integrierter Software-Produktlinien-Spezifikationen, die in der Sprache Clafer spezifiziert sind, präsentiert. Die Sprache Clafer bietet sowohl Sprachmittel zur Charakterisierung der Laufzeitvariabilität eines Systems als auch der rekonfigurierbaren Bestandteile der Systemarchitektur sowie komplexer Abhängigkeiten. In Clafer-Spezifikationen werde hierzu Sprachkonstrukte aus UML-Klassendiagrammen und Meta-Modellierungssprachen mit Feature-orientierten Modellierungstechniken zusammen mit Constraints in Prädikatenlogik erster Stufe kombiniert. Durch die beträchtliche Ausdrucksstärke neigen derartige integrierte Produktlinien-Spezifikationen in der Praxis dazu, sehr komplex zu werden. Sie sind daher äußerst anfällig für Spezifikationsfehler in Form von Inkonsistenzen oder Entwurfsschwächen in Form von Anomalien. Inkonsistenzen und Anomalien müssen jedoch möglichst früh im Entwurfsprozess erkannt und behoben werden, um Systemausfälle zur Laufzeit zu vermeiden. Aus diesem Grund ist die in dieser Arbeit präsentierte statische Analysetechnik zur automatisierten Validierung integrierter Software-Produktlinien-Spezifikationen unabdingbar.

Im Folgenden fassen wir die Ziele sowie die präsentierten wissenschaftlichen Beiträge zusammen und beschreiben die sich aus dieser Arbeit ergebenden zukünftigen Forschungsfelder.

9.1 ZUSAMMENFASSUNG

In dieser Arbeit wurde zunächst anhand des Beispiels eines selbst-adaptiven Kommunikationssystems die Spezifikation struktureller Konsistenzeigenschaften in der Sprache Clafer vorgestellt (**WB 1**). Hierbei wurden potentiell mehrfach instanziierbare Konfigurationsoptionen im Problemraum und Komponenten der Systemarchitektur des Lösungsraums als Clafer-Definitionen der integrierten Produktlinien-Spezifikation beschrieben. Die Clafer-Definitionen wurden hierbei durch Vererbungs-, Kompositions- und Referenzrelationen zueinander in Bezug gesetzt. Abhängigkeiten zwischen Problem- und Lösungsraum sowie Abhängigkeiten rekonfigurierbarer System-Features und durch den Systemkontext gebundene Kontext-Features, wurden in Form von Constraints in Prädikatenlogik erster Stufe formuliert. So konnten sowohl komplexe Attribut-Constraints als auch lokale

instanzbezogene Einschränkungen spezifiziert werden. Es zeigte sich, dass die Sprache Clafer eine geeignete Repräsentation zur ganzheitlichen Spezifikation struktureller Konsistenzeigenschaften selbst-adaptiver Systeme darstellt.

Ausgehend von existierenden Anomalietypen, die in Spezifikationen des Problemraums auftreten, wurden neuartige Anomalietypen vorgestellt, die in Clafer-Spezifikationen auftreten können (**WB 2**). Es wurden die Anomalien vom Typ (i) *tote Clafer-Definition*, (ii) *Kern-Clafer-Definition*, (iii) *falsche Intervallunter-* und *Obergrenze* sowie (iv) *lückenhafter Wertebereich* von Multiplizitätsintervallen charakterisiert, die unerkannt potentiell zu schwerwiegenden Problemen zur Laufzeit des Systems führen und daher zur Entwurfszeit durch automatisierte Analyseverfahren identifiziert werden müssen.

Eine Clafer-Spezifikation kann aufgrund unbeschränkter Multiplizitätsintervalle oder reellwertiger Attribute potentiell eine beliebig große Anzahl von Instanzen repräsentieren. Existierende Verfahren zur Konsistenzprüfung von Clafer-Spezifikationen codieren das zugrundeliegende Analyseproblem entweder in der Spezifikationssprache Alloy oder mittels Constraint-Programming-Techniken. Beide Repräsentationen erfordern jedoch, dass der Suchraum, in dem nach Spezifikationsinstanzen gesucht wird, manuell oder heuristisch eingeschränkt wird. Aus diesem Grund sind existierende Verfahren nicht vollständig. Zur vollständigen Analyse von Clafer-Spezifikationen wurde in dieser Arbeit eine Kernsprache präsentiert, welche Clafer-Spezifikationen auf einen entscheidbaren Kern reduzieren, für den unser Analyseverfahren vollständig und korrekt ist (**WB 3**). Für jede Clafer-Spezifikation, die in der Kernsprache vorliegt und die aufgrund unbeschränkter Multiplizitätsintervalle potentiell beliebig viele Instanzen aufweist, kann (falls der Suchraum beschränkt ist) die Grenze des Suchraums identifiziert werden. Falls der Suchraum unbeschränkt ist, lässt sich dies für Spezifikationen, die in der Kernsprache vorliegen, ebenfalls feststellen. Das in dieser Arbeit präsentierte Analyseverfahren lässt sich zudem komplementär zu existierenden (nicht-vollständigen) Analyseverfahren einsetzen, um Grenzen des Suchraums für Konsistenzprüfungen systematisch zu ermitteln. Darüber hinaus wurde in dieser Arbeit eine erweiterte Kernsprache eingeführt, die im Sprachumfang gegenüber der Kernsprache ausdrucksstärker ist und welche die korrekte Analyse von Clafer-Spezifikationen ermöglicht (**WB 4**).

Weiterhin wurde in dieser Arbeit eine neuartige symbolische Multimengen-Repräsentation von Clafer-Spezifikationen eingeführt, die für die erweiterte Kernsprache Abschätzungen für Auswirkungen von Constraints auf Instanzmengen von Clafer-Definitionen zulässt (**WB 5**). Zur Transformation einer Clafer-Spezifikation in die Multimengen-Repräsentation wurde der Flattening-Algorithmus vorgestellt, der Vererbungshierarchien in Clafer-Spezifikation abflacht, sodass abstrakte Clafer-Definitionen semantikerhaltend durch konkrete Clafer-Definitionen in Constraints ersetzt werden können. Darüber hinaus wurden für sämtliche Sprachelemente von Clafer-Spezifikationen der erweiterten Kernsprache Übersetzungen in die Multimengen-Repräsentation präsentiert. Die Multimengen-Repräsentation ermöglicht es, auch nicht-reduzierte Spezifikationen in Bezug auf die zuvor charakterisierten Anomalietypen zu analysieren. Falls keine Instanz für eine

Spezifikation (die auf die erweiterte Kernsprache reduziert wurde) gefunden wird, kann geschlossen werden, dass die ursprüngliche (nicht-reduzierte) Spezifikation inkonsistent ist. Durch eine experimentelle Evaluation unseres Analyseverfahrens hat sich gezeigt, dass unser Analyseverfahren in der Lage ist, eine beträchtliche Anzahl von Anomalien in praxisrelevanten Spezifikationen zu erkennen, die größtenteils durch konkurrierende Ansätze nicht erkannt werden.

Zur effizienten Analyse von Clafer-Spezifikationen wurde eine Formulierung der Analyseziele sowie der zugrundeliegenden Multimengen-Repräsentation als gemischt-ganzzahliges lineares Optimierungsproblem präsentiert (**WB 6**). Hierzu wurden zunächst Bedingungen der Multimengen-Repräsentation in ein erweitertes Optimierungsproblem als Zwischenrepräsentation übersetzt, das aussagenlogische Verknüpfungen von Constraints und Entscheidungsvariablen zulässt. Es wurde ein Verfahren vorgestellt, das in einem zweiten Schritt aussagenlogische Ausdrücke in ein reguläres mathematisches Optimierungsproblem transformiert und die Anwendung industrieerprobter Standard-Lösungsverfahren erlaubt. Die Repräsentation als mathematisches Optimierungsproblem ermöglicht zudem eine effiziente Handhabung von ganzzahligen und reellwertigen Attributen. Die experimentelle Evaluation hat gezeigt, dass für die Analyse ganzzahliger und reellwertiger Attribute in Clafer-Spezifikationen keine signifikanten Effizienzunterschiede existieren. Dies stellt eine wesentliche Verbesserung gegenüber existierenden Verfahren dar, die nur beschränkte Unterstützung für die Analyse von Spezifikationen mit ganzzahligen und keine Unterstützung für die Analyse reellwertiger Attribute bieten.

Die Evaluation hat gezeigt, dass durch die Abflachung der Vererbungshierarchie und insbesondere durch Abschätzungen von Pfadausdrücken der Zuwachs der Problemgrößen des zugrundeliegenden mathematischen Optimierungsproblems mit zunehmenden Spezifikationen überproportional ist. Weiterhin konnte durch die Evaluation gezeigt werden, dass unser Analyseverfahren für praxisrelevante Spezifikationsgrößen mit vertretbarem Rechenaufwand anwendbar ist und darüber hinaus erhebliche Verbesserungen der Laufzeiteffizienz gegenüber existierenden Verfahren zur Konsistenzprüfung von Clafer-Spezifikation aufweist.

9.2 AUSBLICK

Im Folgenden diskutieren wir zukünftige Forschungsfelder, die sich aus dieser Arbeit ergeben.

DYNAMISCHE BINDUNGSZEITEN: In Abschnitt 2.3.1 wurde als wesentliche Kernerweiterung des Entwicklungsprozesses bei DSPLs gegenüber SPLs die Möglichkeit der Spezifikation von Bindungszeiten genannt. Eine Reihe von Arbeiten existieren, um Einschränkungen zwischen Bindungszeiten zu spezifizieren und zu analysieren [29, 46, 126]. Hierbei werden jedoch entweder traditionelle Feature-Modelle mit ausschließlich bi-

nären Features oder attributierte Feature-Modelle mit eingeschränkten Wertedomänen betrachtet.

Die Sprache Clafer bietet zwar keine explizite Möglichkeit, um Bindungszeiten von Konfigurationsoptionen zu spezifizieren, es ist aber denkbar, durch zusätzliche Strukturen von Hilfs-Clafer-Definitionen Abhängigkeiten zwischen Bindungszeiten sowie Konfigurationsoptionen des Problemraums zu repräsentieren. Da unser Analyseverfahren die vollständige und korrekte Analyse von Spezifikationen mit potentiell unbeschränkten Wertebereichen zulässt (soweit diese in der Kernsprache vorliegen), würde dies eine Vielzahl weiterer Analysen ermöglichen und eine Verbesserung gegenüber existierenden Verfahren darstellen.

UNTERSTÜTZUNG WEITERER ANALYSEZIELE: In Abschnitt 8.1 wurde eine Reihe von Analysetechniken für Problemraumspezifikationen genannt, die zusätzlich zu identifizierten Anomalien auch Erklärungen für Anomalieursachen bzw. Vorschläge zur Beseitigung von Anomalien liefern. Unser Analyseverfahren ist in der Lage, für ein Multiplizitätsintervall Aussagen über ungültige Wertebereiche zu liefern. Es wäre darüber hinaus wünschenswert, im Falle von inkonsistenten Spezifikationen Erklärungshilfen zu Constraints oder Multiplizitätsintervallen zu liefern, die für die Inkonsistenz ursächlich sind. Für Erfüllbarkeitsprobleme wurden hierzu Verfahren zur Berechnung des minimalen unerfüllbaren Kerns vorgeschlagen [95]. Für MILPs könnten zu diesem Zweck beispielsweise Sensitivitätsanalysen eingesetzt werden [48], die durch Stabilitätsbetrachtungen von Lösungen gemischt-ganzzahliger linearer Optimierungsprobleme für Ursachen von Inkonsistenzen liefern könnten.

Darüber hinaus wurden in Abschnitt 8.1 Verfahren genannt, die weitere Spezifikationseigenschaften wie beispielsweise redundante Constraints identifizieren können. Unser Analyseverfahren ermöglicht zwar den Suchraum für die Instanzsuche automatisiert einzuschränken, die Identifikation von Constraints, die den Suchraum nicht einschränken, ist hingegen nicht möglich, in Zukunft jedoch wünschenswert.

ERWEITERUNG DES ANALYSIERBAREN SPRACHUMFANGS: Kürzlich wurden Erweiterungen für die Sprache Clafer vorgestellt, die neben der strukturellen Beschreibung von Feature-Modellen und Komponentenmodellen auch die Spezifikation von Verhalten in Form von Automatenmodellen ermöglichen [77]. Hierfür werden Zustände in Form von Clafer-Definitionen spezifiziert und durch zusätzliche temporale Constraints zueinander in Beziehung gesetzt. Die so erweiterte Sprache Clafer lässt die Spezifikation von strukturellen und verhaltensorientierten Aspekten in einer integrierten Repräsentation zu. Für die Analyse derartiger integrierter Software-Produktlinien-Spezifikationen wurden bisher keine Analyseverfahren vorgestellt. Dies stellt somit ein vielversprechendes zukünftiges Forschungsfeld dar.

LITERATURVERZEICHNIS

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*, S. 304–307, 1999. URL: https://doi.org/10.1007/3-540-48157-5_29. (Zitiert auf Seite 37.)

[2] Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, and Jean Paul Rigault. Modeling context and dynamic adaptations with feature models. In *CEUR Workshop Proceedings*, volume 509, S. 89–98, 2009. URL: https://hal.archives-ouvertes.fr/hal-00419990. (Zitiert auf Seite 37.)

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN: 0321486811 . (Zitiert auf Seite 176.)

[4] Bastian Alt, Markus Weckesser, Christian Becker, Matthias Hollick, Heinz Koeppel, Sounak Kar, Anja Klein, Robin Klose, Roland Kluge, Boris Koldehofe, Wasiur R. KhudaBukhsh, Manisha Luthra, Mahdi Mousavi, Max Mühlhäuser, Martin Pfannemüller, Amr Rizk, Andy Schürr, and Ralf Steinmetz. Transitions: A Protocol-Independent View of the Future Internet. *Proceedings of the IEEE*, S. 1–13, 2019. (Zitiert auf Seite 14.)

[5] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On Challenges of Model Transformation from UML to Alloy. *Software & Systems Modeling*, 9(1):69–86, 2010. (Zitiert auf Seite 199.)

[6] Michal Antkiewicz, Kacper Bak, Alexandr Murashkin, Rafael Olaechea, Jia Hui (Jimmy) Liang, and Krzysztof Czarnecki. Clafer tools for product line engineering. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, SPLC '13 Workshops, S. 130–135, New York, NY, USA, 2013. ACM. (Zitiert auf den Seiten 8, 9, 56, 57, 179 und 202.)

[7] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013. ISBN: 978-3-642-37520-0 . URL: http://dx.doi.org/10.1007/978-3-642-37521-7. (Zitiert auf den Seiten 4, 20, 21 und 22.)

[8] Matthias Baaz, Uwe Egly, and Alexander Leitsch. Normal form transformations. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, S. 273–333. Elsevier and MIT Press, 2001. (Zitiert auf Seite 137.)

[9] Kacper Bak, Krzysztof Czarnecki, and Andrzej Wasowski. Feature and meta-models in clafer: Mixed, specialized, and coupled. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, S. 102–122. Springer, 2010. URL: https://doi.org/10.1007/978-3-642-19440-5_7. (Zitiert auf den Seiten 7, 48 und 53.)

[10] Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. Clafer: unifying class and feature modeling. *Software and System Modeling*, 15(3):811–845, 2016. URL: https://doi.org/10.1007/s10270-014-0441-1. (Zitiert auf den Seiten 5, 48, 83, 183 und 184.)

[11] Mira Balaban and Azzam Maraee. Consistency of UML class diagrams with hierarchy constraints. In Opher Etzion, Tsvi Kuflik, and Amihai Motro, editors, *Next Generation Information Technologies and Systems, 6th International Workshop, NGITS 2006, Kibbutz Shefayim, Israel, July 4-6, 2006, Proceedings*, volume 4032 of *Lecture Notes in Computer Science*, S. 71–82. Springer, 2006. URL: https://doi.org/10.1007/11780991_7. (Zitiert auf Seite 200.)

[12] Mira Balaban and Azzam Maraee. Finite satisfiability of UML class diagrams with constrained class hierarchy. *ACM Trans. Softw. Eng. Methodol.*, 22(3):24:1–24:42, 2013. URL: https://doi.org/10.1145/2491509.2491518. (Zitiert auf Seite 200.)

[13] Mira Balaban and Azzam Maraee. Simplification and Correctness of UML Class Diagrams - Focusing on Multiplicity and Aggregation/Composition Constraints. In *MODELS 2013*, S. 454–470, 2013. (Zitiert auf Seite 200.)

[14] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Trans. Software Eng.*, 41(5):507–525, 2015. URL: https://doi.org/10.1109/TSE.2014.2372785. (Zitiert auf Seite 178.)

[15] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking.*, S. 305–343. Springer, 2018. URL: https://doi.org/10.1007/978-3-319-10575-8_11. (Zitiert auf Seite 202.)

[16] Don S. Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines, 9th International Conference,*

SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings, volume 3714 of *Lecture Notes in Computer Science*, S. 7–20. Springer, 2005. URL: https://doi.org/10.1007/11554844_3. (Zitiert auf den Seiten 8, 24, 33, 195 und 196.)

[17] Don S. Batory, David Benavides, and Antonio Ruiz Cortés. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49(12):45–47, 2006. URL: https://doi.org/10.1145/1183236.1183264. (Zitiert auf Seite 33.)

[18] Evelyn Martin Lansdowne Beale and John A Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR*, 69(447-454), 1970. (Zitiert auf Seite 168.)

[19] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, S. 491–503, 2005. URL: https://doi.org/10.1007/11431855_34. (Zitiert auf den Seiten 8, 33, 34 und 195.)

[20] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010. URL: https://doi.org/10.1016/j.is.2010.01.001. (Zitiert auf den Seiten 7, 33, 34, 35 und 196.)

[21] David Benavides, Sergio Segura, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. Using java CSP solvers in the automated analyses of feature models. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, S. 399–408, 2005. URL: http://dx.doi.org/10.1007/11877028_16. (Zitiert auf Seite 33.)

[22] Nelly Bencomo, Svein O. Hallsteinsen, and Eduardo Santana de Almeida. A view of the dynamic software product line landscape. *IEEE Computer*, 45(10):36–41, 2012. URL: https://doi.org/10.1109/MC.2012.292. (Zitiert auf den Seiten 3, 18 und 38.)

[23] Nelly Bencomo, Peter Sawyer, Gordon S. Blair, and Paul Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*, S. 23–32, 2008. (Zitiert auf den Seiten 3, 4 und 36.)

[24] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artif. Intell.*, 168(1-2):70–118, 2005. URL: https://doi.org/10.1016/j.artint.2005.05.003. (Zitiert auf Seite 199.)

[25] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: From problem to solution space. In *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, SAICSIT '05, S. 182–191, Republic of South Africa, 2005. South African Institute for Computer Scientists and Information Technologists. URL: http://dl.acm.org/citation.cfm?id=1145675.1145695. (Zitiert auf Seite 48.)

[26] Faouzi Boufarès and Hachemi Bennaceur. Consistency problems in er-schemas for database systems. *Inf. Sci.*, 163(4):263–274, 2004. URL: https://doi.org/10.1016/j.ins.2003.06.015. (Zitiert auf Seite 202.)

[27] Mark Brodie, Sheng Ma, Guy M. Lohman, Laurent Mignet, Natwar Modani, Mark Wilding, Jon Champlin, and Peter Sohn. Quickly finding known software problems via automated symptom matching. In *Second International Conference on Autonomic Computing (ICAC 2005), 13-16 June 2005, Seattle, WA, USA*, S. 101–110, 2005. URL: https://doi.org/10.1109/ICAC.2005.49. (Zitiert auf Seite 3.)

[28] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, S. 240–254, 2012. URL: https://doi.org/10.1007/978-3-642-28872-2_17. (Zitiert auf Seite 3.)

[29] Johannes Bürdek, Sascha Lity, Malte Lochau, Markus Berens, Ursula Goltz, and Andy Schürr. Staged configuration of dynamic software product lines with complex binding time constraints. In Philippe Collet, Andrzej Wasowski, and Thorsten Weyer, editors, *The Eighth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '14, Sophia Antipolis, France, January 22-24, 2014*, S. 16:1–16:8. ACM, 2014. URL: https://doi.org/10.1145/2556624.2556627. (Zitiert auf den Seiten 8, 34, 36, 198 und 205.)

[30] Sven Burmester, Holger Giese, Eckehard Münch, Oliver Oberschelp, Florian Klein, and Peter Scheideler. Tool support for the design of self-optimizing mechatronic multi-agent systems. *STTT*, 10(3):207–222, 2008. URL: https://doi.org/10.1007/s10009-008-0067-0. (Zitiert auf Seite 3.)

[31] Jordi Cabot, Robert Clarisó, and Daniel Riera. Umltocsp: a tool for the formal verification of UML/OCL models using constraint programming. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, S. 547–548. ACM, 2007. URL: https://doi.org/10.1145/1321631.1321737. (Zitiert auf Seite 201.)

[32] Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014. URL: https://doi.org/10.1016/j.jss.2014.03.023. (Zitiert auf Seite 201.)

[33] Jordi Cabot and Martin Gogolla. Object constraint language (OCL): A definitive guide. In Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio, editors, *Formal Methods for Model-Driven Engineering - 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*, volume 7320 of *Lecture Notes in Computer Science*, S. 58–90. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-30982-3_3. (Zitiert auf den Seiten 43 und 72.)

[34] Marco Cadoli, Diego Calvanese, Giuseppe De Giacomo, and Toni Mancini. Finite Model Reasoning on UML Class Diagrams via Constraint Programming. In *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*, S. 36–47, 2007. (Zitiert auf Seite 199.)

[35] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A formal framework for reasoning on UML class diagrams. In Mohand-Said Hacid, Zbigniew W. Ras, Djamel A. Zighed, and Yves Kodratoff, editors, *Foundations of Intelligent Systems, 13th International Symposium, ISMIS 2002, Lyon, France, June 27-29, 2002, Proceedings*, volume 2366 of *Lecture Notes in Computer Science*, S. 503–513. Springer, 2002. URL: https://doi.org/10.1007/3-540-48050-1_54. (Zitiert auf Seite 199.)

[36] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz Cortés, and Mike Hinchey. An overview of dynamic software product line architectures and techniques: Observations from research and industry. *Journal of Systems and Software*, 91:3–23, 2014. URL: https://doi.org/10.1016/j.jss.2013.12.038. (Zitiert auf den Seiten 6 und 36.)

[37] Junjie Chen, Wenxiang Hu, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. An empirical comparison of compiler testing techniques. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, S. 180–190, 2016. URL: https://doi.org/10.1145/2884781.2884878. (Zitiert auf Seite 178.)

[38] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software

engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, S. 1–26, 2009. URL: https://doi.org/10.1007/978-3-642-02161-9_1. (Zitiert auf Seite 5.)

[39] Andreas Classen, Arnaud Hubaux, Franciscus Sanen, Eddy Truyen, Jorge Vallejos, Pascal Costanza, Wolfgang De Meuter, Patrick Heymans, and Wouter Joosen. Modelling variability in self-adaptive systems: Towards a research agenda. In *Proc. of the 1st Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering held as part of GPCE08*, 2008. (Zitiert auf Seite 36.)

[40] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Eng. Addison-Wesley, 2001. (Zitiert auf den Seiten 4 und 19.)

[41] Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, S. 472–481. IEEE Computer Society, 2013. URL: https://doi.org/10.1109/ICSE.2013.6606593. (Zitiert auf den Seiten 6, 8, 34, 73 und 197.)

[42] Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In Robert Glück and Michael R. Lowry, editors, *Generative Programming and Component Engineering, 4th International Conference, GPCE 2005, Tallinn, Estonia, September 29 - October 1, 2005, Proceedings*, volume 3676 of *Lecture Notes in Computer Science*, S. 422–437. Springer, 2005. URL: https://doi.org/10.1007/11561347_28. (Zitiert auf Seite 5.)

[43] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. ISBN: 0-201-30977-7 . (Zitiert auf den Seiten 4, 7, 21, 22 und 27.)

[44] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration using feature models. In Robert L. Nord, editor, *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, S. 266–283. Springer, 2004. URL: https://doi.org/10.1007/978-3-540-28630-1_17. (Zitiert auf den Seiten 29, 30, 32, 36 und 196.)

[45] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. URL: https://doi.org/10.1002/spip.213. (Zitiert auf den Seiten 6, 8, 29, 51 und 196.)

[46] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software*

Process: Improvement and Practice, 10(2):143–169, 2005. URL: http://dx.doi.org/10.1002/spip.225. (Zitiert auf den Seiten 36, 48 und 205.)

[47] Ferruccio Damiani and Ina Schaefer. Dynamic delta-oriented programming. In Ina Schaefer, Isabel John, and Klaus Schmid, editors, *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Workshop Proceedings (Volume 2)*, S. 34. ACM, 2011. URL: https://doi.org/10.1145/2019136.2019175. (Zitiert auf Seite 5.)

[48] M. W. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. *Operations Research*, 48(4):623–634, 2000. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.48.4.623.12420. (Zitiert auf Seite 206.)

[49] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, S. 502–518. Springer, 2003. URL: https://doi.org/10.1007/978-3-540-24605-3_37. (Zitiert auf Seite 179.)

[50] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, S. 7–16. ACM, 2010. URL: https://doi.org/10.1145/1882291.1882296. (Zitiert auf Seite 39.)

[51] Andreas Falkner, Ingo Feinerer, Gernot Salzer, and Gottfried Schenner. Computing product configurations via UML and integer linear programming. *International Journal of Mass Customisation*, 3(4):351–367, 2010. (Zitiert auf Seite 200.)

[52] Ingo Feinerer and Gernot Salzer. Numeric semantics of class diagrams with multiplicity and uniqueness constraints. *Software and System Modeling*, 13(3):1167–1187, 2014. URL: https://doi.org/10.1007/s10270-012-0294-4. (Zitiert auf Seite 200.)

[53] Ingo Feinerer, Gernot Salzer, and Tanja Sisel. Reducing multiplicities in class diagrams. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, S. 379–393. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-24485-8_27. (Zitiert auf Seite 200.)

[54] Alexander Felfernig, David Benavides, José A. Galindo, and Florian Reinfrank. Towards anomaly explanation in feature models. In Michel Aldanondo and Andreas A. Falkner, editors, *Proceedings of the 15th International Configuration Workshop,*

Vienna, Austria, August 29-30, 2013., volume 1128 of *CEUR Workshop Proceedings*, S. 117–124. CEUR-WS.org, 2013. URL: http://ceur-ws.org/Vol-1128/paper16.pdf. (Zitiert auf Seite 196.)

[55] Paula Fernandes, Cláudia Werner, and Eldânae Teixeira. An approach for feature modeling of context-aware software product line. *J. UCS*, 17(5):807–829, 2011. URL: https://doi.org/10.3217/jucs-017-05-0807. (Zitiert auf Seite 37.)

[56] Sylvain Frey, François Huguet, Cédric Mivielle, David Menga, Ada Diaconescu, and Isabelle M. Demeure. Scenarios for an autonomic micro smart grid. In *SMARTGREENS 2012 - Proceedings of the 1st International Conference on Smart Grids and Green IT Systems, Porto, Portugal, 19 - 20 April, 2012*, S. 137–140, 2012. (Zitiert auf Seite 3.)

[57] José A. Galindo, David Benavides, Pablo Trinidad, Antonio-Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés. Automated analysis of feature models: Quo vadis? *Computing*, Aug 2018. URL: https://doi.org/10.1007/s00607-018-0646-1. (Zitiert auf Seite 7.)

[58] Elena Gaura, Michael Allen, Lewis Girod, James Brusey, and Geoffrey Challen. *Wireless Sensor Networks: Deployments and Design Frameworks*. Springer, 1 edition, 2016. ISBN: 978-1-4419-5833-4 . URL: https://doi.org/10.1007/978-1-4419-5834-1. (Zitiert auf Seite 14.)

[59] Holger Giese and Wilhelm Schäfer. Model-driven development of safe self-optimizing mechatronic systems with mechatronicuml. In Javier Cámara, Rogério de Lemos, Carlo Ghezzi, and Antónia Lopes, editors, *Assurances for Self-Adaptive Systems - Principles, Models, and Techniques*, volume 7740 of *Lecture Notes in Computer Science*, S. 152–186. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-36249-1_6. (Zitiert auf Seite 3.)

[60] Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A uml-based specification environment for validating UML and OCL. *Sci. Comput. Program.*, 69(1-3):27–34, 2007. URL: https://doi.org/10.1016/j.scico.2007.01.013. (Zitiert auf Seite 201.)

[61] Hassan Gomaa. *Designing software product lines with UML - from use cases to pattern-based software architectures*. ACM, 2005. ISBN: 978-0-201-77595-2 . (Zitiert auf Seite 48.)

[62] Marco Gramaglia, Óscar Trullols-Cruces, Diala Naboulsi, Marco Fiore, and María Calderón. Mobility and connectivity in highway vehicular networks: A case study in madrid. *Computer Communications*, 78:28–44, 2016. URL: https://doi.org/10.1016/j.comcom.2015.10.014. (Zitiert auf Seite 14.)

[63] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2015. URL: http://www.gurobi.com. (Zitiert auf den Seiten 178 und 190.)

[64] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008. URL: http://dx.doi.org/10.1109/MC.2008.123. (Zitiert auf den Seiten 4 und 36.)

[65] Svein O. Hallsteinsen, Erlend Stav, Arnor Solberg, and Jacqueline Floch. Using product line techniques to build adaptive systems. In *Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings*, S. 141–150, 2006. URL: https://doi.org/10.1109/SPLINE.2006.1691586. (Zitiert auf Seite 36.)

[66] Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings*, S. 12–21, 2008. URL: https://doi.org/10.1109/SPLC.2008.15. (Zitiert auf den Seiten 4, 6 und 37.)

[67] Øystein Haugen, Andrzej Wasowski, and Krzysztof Czarnecki. CVL: common variability language. In Tomoji Kishi, Stan Jarzabek, and Stefania Gnesi, editors, *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*, S. 277. ACM, 2013. URL: https://doi.org/10.1145/2491627.2493899. (Zitiert auf den Seiten 5, 7 und 48.)

[68] Florian Heidenreich, Jan Kopcsek, and Christian Wende. Featuremapper: mapping features to models. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume*, S. 943–944. ACM, 2008. URL: https://doi.org/10.1145/1370175.1370199. (Zitiert auf Seite 48.)

[69] Patrick Heymans, Pierre-Yves Schobbens, Jean-Christophe Trigaux, Yves Bontemps, Raimundas Matulevicius, and Andreas Classen. Evaluating formal properties of feature diagram languages. *IET Software*, 2(3):281–302, 2008. URL: https://doi.org/10.1049/iet-sen:20070055. (Zitiert auf den Seiten 34 und 195.)

[70] Michael G. Hinchey and Roy Sterritt. Self-managing software. *IEEE Computer*, 39(2):107–109, 2006. URL: https://doi.org/10.1109/MC.2006.69. (Zitiert auf Seite 3.)

[71] Mike Hinchey, Sooyong Park, and Klaus Schmid. Building dynamic software product lines. *IEEE Computer*, 45(10):22–26, 2012. URL: http://dx.doi.org/10.1109/MC.2012.332. (Zitiert auf den Seiten 37 und 38.)

[72] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28, 2008. URL: http://doi.acm.org/10.1145/1380584.1380585. (Zitiert auf den Seiten 3, 4 und 18.)

[73] Heinrich Hußmann, Birgit Demuth, and Frank Finger. Modular architecture for a toolset supporting OCL. *Sci. Comput. Program.*, 44(1):51–69, 2002. URL: https://doi.org/10.1016/S0167-6423(02)00032-1. (Zitiert auf Seite 201.)

[74] IEEE STD 1044™–2009. IEEE Standard Classification for Software Anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, Jan 2010. (Zitiert auf den Seiten 7, 35 und 58.)

[75] ISO Central Secretary. Information technology – syntactic metalanguage – extended bnf. Standard ISO/IEC 14977:1996(E), International Organization for Standardization, Geneva, CH, 1996. URL: https://www.iso.org/standard/62711.html. (Zitiert auf den Seiten 69 und 228.)

[76] Daniel Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006. ISBN: 978-0-262-10114-1 . (Zitiert auf den Seiten 9, 57, 179, 199 und 202.)

[77] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. Clafer: Lightweight modeling of structure, behaviour, and variability. *CoRR*, abs/1807.08576, 2018. URL: http://arxiv.org/abs/1807.08576. (Zitiert auf den Seiten 11 und 206.)

[78] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, 1990. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231. (Zitiert auf den Seiten 4, 7, 21, 22 und 27.)

[79] Kyo Chul Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Software Eng.*, 5:143–168, 1998. URL: https://doi.org/10.1023/A:1018980625587. (Zitiert auf Seite 27.)

[80] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. Mapping extended feature models to constraint logic programming over finite domains. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, S. 286–299, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1885639.1885666. (Zitiert auf den Seiten 8, 27, 28, 34 und 198.)

[81] Ahmet Serkan Karatas, Halit Oguztüzün, and Ali H. Dogru. From extended feature models to constraint logic programming. *Sci. Comput. Program.*, 78(12):2295–2312,

2013. URL: http://dx.doi.org/10.1016/j.scico.2012.06.004. (Zitiert auf den Seiten 28 und 198.)

[82] David Kempe, Jon M. Kleinberg, and Alan J. Demers. Spatial gossip and resource location protocols. *J. ACM*, 51(6):943–967, 2004. URL: http://doi.acm.org/10.1145/1039488.1039491. (Zitiert auf Seite 15.)

[83] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003. URL: https://doi.org/10.1109/MC.2003.1160055. (Zitiert auf den Seiten 3 und 18.)

[84] Eldar Khalilov, Jordan A. Ross, Michal Antkiewicz, Markus Völter, and Krzysztof Czarnecki. Modeling and optimizing automotive electric/electronic (E/E) architectures: Towards making clafer accessible to practitioners. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II*, volume 9953 of *Lecture Notes in Computer Science*, S. 447–464, 2016. URL: https://doi.org/10.1007/978-3-319-47169-3_37. (Zitiert auf Seite 9.)

[85] Samuel Kounev, Peter R. Lewis, Kirstie L. Bellman, Nelly Bencomo, Javier Cámara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey O. Kephart, and Andrea Zisman. The notion of self-aware computing. In Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu, editors, *Self-Aware Computing Systems.*, S. 3–16. Springer International Publishing, 2017. URL: https://doi.org/10.1007/978-3-319-47474-8_1. (Zitiert auf Seite 3.)

[86] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. Explaining anomalies in feature models. In Bernd Fischer and Ina Schaefer, editors, *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2016, Amsterdam, The Netherlands, October 31 - November 1, 2016*, S. 132–143. ACM, 2016. URL: https://doi.org/10.1145/2993236.2993248. (Zitiert auf Seite 196.)

[87] Dean Kramer, Christian Severin Sauer, and Thomas Roth-Berghofer. Towards explanation generation using feature models in software product lines. In Grzegorz J. Nalepa and Joachim Baumeister, editors, *Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9) co-located with the 36th German Conference on Artificial Intelligence (KI2013), Koblenz, Germany, September 17, 2013.*, volume 1070 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013. URL: http://ceur-ws.org/Vol-1070/kese9-02_03.pdf. (Zitiert auf Seite 196.)

[88] Philippe Kruchten, Bran Selic, and Wojtek Kozaczynski. Describing software architecture with UML. In Hausi A. Müller, Mary Jean Harrold, and Wilhelm Schäfer,

editors, *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, S. 715–716. IEEE Computer Society, 2001. URL: https://doi.org/10.1109/ICSE.2001.919174. (Zitiert auf Seite 40.)

[89] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015. URL: https://doi.org/10.1016/j.pmcj.2014.09.009. (Zitiert auf Seite 4.)

[90] Jaejoon Lee and Kyo Chul Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In *Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings*, S. 131–140, 2006. URL: https://doi.org/10.1109/SPLINE.2006.1691585. (Zitiert auf Seite 36.)

[91] Maurizio Lenzerini and Paolo Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Inf. Syst.*, 15(4):453–461, 1990. URL: https://doi.org/10.1016/0306-4379(90)90048-T. (Zitiert auf Seite 202.)

[92] Uwe Lesta, Ina Schaefer, and Tim Winkelmann. Detecting and explaining conflicts in attributed feature models. In Joanne M. Atlee and Stefania Gnesi, editors, *Proceedings 6th Workshop on Formal Methods and Analysis in SPL Engineering, FMSPLE@ETAPS 2015, London, UK, 11 April 2015.*, volume 182 of *EPTCS*, S. 31–43, 2015. URL: https://doi.org/10.4204/EPTCS.182.3. (Zitiert auf Seite 197.)

[93] Jia Hui (Jimmy) Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. Sat-based analysis of large real-world feature models is easy. In Douglas C. Schmidt, editor, *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, S. 91–100. ACM, 2015. URL: https://doi.org/10.1145/2791060.2791070. (Zitiert auf den Seiten 8 und 33.)

[94] Malte Lochau, Johannes Bürdek, Stefan Hölzle, and Andy Schürr. Specification and automated validation of staged reconfiguration processes for dynamic software product lines. *Software and System Modeling*, 16(1):125–152, 2017. URL: https://doi.org/10.1007/s10270-015-0470-4. (Zitiert auf Seite 198.)

[95] Inês Lynce and João P. Marques Silva. On computing minimum unsatisfiable cores. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. URL: http://www.satisfiability.org/SAT04/programme/110.pdf. (Zitiert auf Seite 206.)

[96] Mike Mannion. Using first-order logic for product line model validation. In Gary J. Chastek, editor, *Software Product Lines, Second International Conference, SPLC 2, San*

Diego, CA, USA, August 19-22, 2002, Proceedings, volume 2379 of *Lecture Notes in Computer Science*, S. 176–187. Springer, 2002. URL: https://doi.org/10.1007/3-540-45652-X_11. (Zitiert auf Seite 25.)

[97] Mike Mannion and Javier Cámara. Theorem proving for product line model verification. In Frank van der Linden, editor, *Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers*, volume 3014 of *Lecture Notes in Computer Science*, S. 211–224. Springer, 2003. URL: https://doi.org/10.1007/978-3-540-24667-1_16. (Zitiert auf Seite 25.)

[98] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Cd2alloy: Class diagrams analysis using alloy revisited. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, S. 592–607. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-24485-8_44. (Zitiert auf Seite 200.)

[99] Azzam Maraee and Mira Balaban. Removing redundancies and deducing equivalences in UML class diagrams. In Jürgen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfrán, editors, *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings*, volume 8767 of *Lecture Notes in Computer Science*, S. 235–251. Springer, 2014. URL: https://doi.org/10.1007/978-3-319-11653-2_15. (Zitiert auf Seite 200.)

[100] Jacopo Mauro, Michael Nieke, Christoph Seidl, and Ingrid Chieh Yu. Context aware reconfiguration in software product lines. In Ina Schaefer, Vander Alves, and Eduardo Santana de Almeida, editors, *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, Salvador, Brazil, January 27 - 29, 2016*, S. 41–48. ACM, 2016. URL: https://doi.org/10.1145/2866614.2866620. (Zitiert auf Seite 37.)

[101] William M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998. URL: http://www.hpl.hp.com/hpjournal/dtj/vol10num1/vol10num1art9.pdf. (Zitiert auf Seite 178.)

[102] Marcílio Mendonça, Andrzej Wasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In Dirk Muthig and John D. McGregor, editors, *Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24-28, 2009, Proceedings*, volume 446 of *ACM International Conference Proceeding Series*, S. 231–240. ACM, 2009. URL: https://dl.acm.org/citation.cfm?id=1753267. (Zitiert auf den Seiten 8, 33 und 195.)

[103] Stephan Mennicke, Malte Lochau, Julia Schroeter, and Tim Winkelmann. Automated verification of feature model configuration processes based on workflow petri

nets. In *18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014*, S. 62–71, 2014. URL: http://doi.acm.org/10.1145/2648511.2648518. (Zitiert auf Seite 198.)

[104] Kim Mens, Rafael Capilla, Nicolás Cardozo, and Bruno Dumas. A taxonomy of context-aware software variability approaches. In *Companion Proceedings of the 15th International Conference on Modularity, Málaga, Spain, March 14 - 18, 2016*, S. 119–124, 2016. URL: http://doi.acm.org/10.1145/2892664.2892684. (Zitiert auf den Seiten 4, 6 und 37.)

[105] Raphaël Michel, Andreas Classen, Arnaud Hubaux, and Quentin Boucher. A formal semantics for feature cardinalities in feature diagrams. In Patrick Heymans, Krzysztof Czarnecki, and Ulrich W. Eisenecker, editors, *Fifth International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium, January 27-29, 2011. Proceedings*, ACM International Conference Proceedings Series, S. 82–89. ACM, 2011. URL: https://doi.org/10.1145/1944892.1944902. (Zitiert auf den Seiten 8, 32, 53 und 197.)

[106] Michael Nieke, Jacopo Mauro, Christoph Seidl, Thomas Thüm, Ingrid Chieh Yu, and Felix Franzke. Anomaly analyses for feature-model evolution. In Eric Van Wyk and Tiark Rompf, editors, *Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2018, Boston, MA, USA, November 5-6, 2018*, S. 188–201. ACM, 2018. URL: https://doi.org/10.1145/3278122.3278123. (Zitiert auf Seite 196.)

[107] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based runtime software evolution. In *Forging New Links, Proceedings of the 1998 International Conference on Software Engineering, ICSE 98, Kyoto, Japan, April 19-25, 1998.*, S. 177–186, 1998. URL: https://doi.org/10.1109/ICSE.1998.671114. (Zitiert auf Seite 4.)

[108] David Lorge Parnas. On the design and development of program families. *IEEE Trans. Software Eng.*, 2(1):1–9, 1976. URL: https://doi.org/10.1109/TSE.1976.233797. (Zitiert auf Seite 19.)

[109] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013. ISBN: 1934356999, 9781934356999 . (Zitiert auf Seite 177.)

[110] Leonardo Passos, Marko Novakovic, Yingfei Xiong, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. A study of non-boolean constraints in variability models of an embedded operating system. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, S. 2:1–2:8, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/2019136.2019139. (Zitiert auf den Seiten 28 und 197.)

[111] Martin Pfannemüller, Christian Krupitzer, Markus Weckesser, and Christian Becker. A dynamic software product line approach for adaptation planning in autonomic computing systems. In Xiaorui Wang, Christopher Stewart, and Hui Lei, editors, *2017 IEEE International Conference on Autonomic Computing, ICAC 2017, Columbus, OH, USA, July 17-21, 2017*, S. 247–254. IEEE Computer Society, 2017. URL: https://doi.org/10.1109/ICAC.2017.18. (Zitiert auf Seite 4.)

[112] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, 2005. ISBN: 978-3-540-24372-4 . URL: http://www.ebook.de/de/product/4437317/guenter_boeckle_klaus_pohl_frank_j_van_der_linden_software_product_line_engineering.html. (Zitiert auf den Seiten 4, 19, 20, 21 und 228.)

[113] Jon Postel. UDP: User datagram protocol. IETF RFC 768, 1980. (Zitiert auf Seite 14.)

[114] Jon Postel. TCP: Transmission control protocol. IETF RFC 793, 1981. (Zitiert auf Seite 14.)

[115] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: http://www.choco-solver.org. (Zitiert auf Seite 179.)

[116] C. Quinton, D. Romero, and L. Duchien. Automated Selection and Configuration of Cloud Environments Using Software Product Lines Principles. In *IEEE Cloud 2014*, S. 144–151, 2014. (Zitiert auf Seite 197.)

[117] Clément Quinton, Andreas Pleuss, Daniel Le Berre, Laurence Duchien, and Goetz Botterweck. Consistency checking for the evolution of cardinality-based feature models. In Stefania Gnesi, Alessandro Fantechi, Patrick Heymans, Julia Rubin, Krzysztof Czarnecki, and Deepak Dhungana, editors, *18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014*, S. 122–131. ACM, 2014. URL: https://doi.org/10.1145/2648511.2648524. (Zitiert auf den Seiten 8, 34 und 197.)

[118] Clément Quinton, Rick Rabiser, Michael Vierhauser, Paul Grünbacher, and Luciano Baresi. Evolution in dynamic software product lines: challenges and perspectives. In Douglas C. Schmidt, editor, *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, S. 126–130. ACM, 2015. URL: https://doi.org/10.1145/2791060.2791101. (Zitiert auf Seite 6.)

[119] Clément Quinton, Daniel Romero, and Laurence Duchien. Cardinality-based feature models with constraints: a pragmatic approach. In Tomoji Kishi, Stan Jarzabek, and Stefania Gnesi, editors, *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*, S. 162–166. ACM, 2013. URL:

https://doi.org/10.1145/2491627.2491638. (Zitiert auf den Seiten 8, 31, 34, 196 und 197.)

[120] Lukman Ab. Rahim and Jon Whittle. A survey of approaches for verifying model transformations. *Software and System Modeling*, 14(2):1003–1028, 2015. URL: https://doi.org/10.1007/s10270-013-0358-0. (Zitiert auf Seite 178.)

[121] R. Raman and I.E. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563 – 578, 1994. URL: http://www.sciencedirect.com/science/article/pii/0098135493E00107. An International Journal of Computer Applications in Chemical Engineering. (Zitiert auf Seite 161.)

[122] Björn Richerzhagen, Dominik Stingl, Ronny Hans, Christian Gross, and Ralf Steinmetz. Bypassing the cloud: Peer-assisted event dissemination for augmented reality games. In *14th IEEE International Conference on Peer-to-Peer Computing, P2P 2014, London, United Kingdom, September 9-11, 2014, Proceedings*, S. 1–10, 2014. URL: https://doi.org/10.1109/P2P.2014.6934296. (Zitiert auf den Seiten 13 und 14.)

[123] Björn Richerzhagen, Dominik Stingl, Ronny Hans, Christian Groß, and Ralf Steinmetz. Bypassing the Cloud: Peer-assisted Event Dissemination for Augmented Reality Games. In *P2P 2014*, S. 1–10, 2014. (Zitiert auf Seite 15.)

[124] Nils Richerzhagen, Dominik Stingl, Björn Richerzhagen, Andreas Mauthe, and Ralf Steinmetz. Adaptive monitoring for mobile networks in challenging environments. In *24th International Conference on Computer Communication and Networks, ICCCN 2015, Las Vegas, NV, USA, August 3-6, 2015*, S. 1–8, 2015. URL: https://doi.org/10.1109/ICCCN.2015.7288371. (Zitiert auf den Seiten 13 und 15.)

[125] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending Feature Diagrams with UML Multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT)*, 2002. (Zitiert auf den Seiten 7, 27, 29 und 196.)

[126] Marko Rosenmüller, Norbert Siegmund, Sven Apel, and Gunter Saake. Flexible feature binding in software product lines. *Automated Software Engineering*, 18(2):163–197, 2011. URL: http://dx.doi.org/10.1007/s10515-011-0080-5. (Zitiert auf den Seiten 36 und 205.)

[127] Jordan A. Ross, Alexandr Murashkin, Jia Hui Liang, Michał Antkiewicz, and Krzysztof Czarnecki. Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems. *Software & Systems Modeling*, S. 1–29, 2017. (Zitiert auf den Seiten 183 und 184.)

[128] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004. ISBN: 0321245628 . (Zitiert auf den Seiten 4, 40, 44 und 201.)

[129] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, may 2009. URL: https://doi.org/10.1145/1516533.1516538. (Zitiert auf den Seiten 3 und 13.)

[130] Karsten Saller, Malte Lochau, and Ingo Reimund. Context-aware dspls: Model-based runtime adaptation for resource-constrained systems. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, SPLC '13 Workshops, S. 106–113, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2499777.2500716. (Zitiert auf den Seiten 4, 6 und 37.)

[131] Karsten Saller, Sebastian Oster, Andy Schürr, Julia Schroeter, and Malte Lochau. Reducing feature models to improve runtime adaptivity on resource limited devices. In *16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012, Volume 2*, S. 135–142, 2012. URL: http://doi.acm.org/10.1145/2364412.2364435. (Zitiert auf Seite 37.)

[132] Martin WP Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994. (Zitiert auf Seite 202.)

[133] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Fifth International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium, January 27-29, 2011. Proceedings*, S. 119–126, 2011. URL: http://doi.acm.org/10.1145/1944892.1944907. (Zitiert auf Seite 21.)

[134] Thomas Schnabel, Markus Weckesser, Roland Kluge, Malte Lochau, and Andy Schürr. Cardygan: Tool support for cardinality-based feature models. In Ina Schaefer, Vander Alves, and Eduardo Santana de Almeida, editors, *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, Salvador, Brazil, January 27 - 29, 2016*, S. 33–40. ACM, 2016. URL: https://doi.org/10.1145/2866614.2866619. (Zitiert auf Seite 177.)

[135] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA*, S. 136–145. IEEE Computer Society, 2006. URL: https://doi.org/10.1109/RE.2006.23. (Zitiert auf den Seiten 34 und 195.)

[136] Seyyed Madasar Ali Shah, Kyriakos Anastasakis, and Behzad Bordbar. From UML to alloy and back again. In Sudipto Ghosh, editor, *Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*, volume 6002 of *Lecture Notes in Computer Science*, S.

158–171. Springer, 2009. URL: https://doi.org/10.1007/978-3-642-12261-3_16. (Zitiert auf Seite 199.)

[137] Norbert Siegmund, Martin Kuhlemann, Marko Rosenmüller, Christian Kästner, and Gunter Saake. Integrated product line model for semi-automated product derivation using non-functional properties. In Patrick Heymans, Kyo Chul Kang, Andreas Metzger, and Klaus Pohl, editors, *Second International Workshop on Variability Modelling of Software-Intensive Systems, Universität Duisburg-Essen, Germany, January 16-18, 2008, Proceedings*, ICB Research Report, S. 25–32, 2008. URL: http://www.vamos-workshop.net/proceedings/VaMoS_2008_Proceedings.pdf. (Zitiert auf Seite 48.)

[138] Sandra Slaughter, Donald E. Harter, and Mayuram S. Krishnan. Evaluating the cost of software quality. *Commun. ACM*, 41(8):67–73, 1998. URL: https://doi.org/10.1145/280324.280335. (Zitiert auf Seite 5.)

[139] Periklis Sochos, Ilka Philippow, and Matthias Riebisch. Feature-oriented development of software product lines: Mapping feature models to the architecture. In Mathias Weske and Peter Liggesmeyer, editors, *Object-Oriented and Internet-Based Technologies, 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a NetworkedWorld, Net.ObjectDays 2004, Erfurt, Germany, September 27-30, 2004, Proceedings*, volume 3263 of *Lecture Notes in Computer Science*, S. 138–152. Springer, 2004. URL: https://doi.org/10.1007/978-3-540-30196-7_11. (Zitiert auf Seite 48.)

[140] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005. URL: http://dx.doi.org/10.1002/spe.652. (Zitiert auf Seite 36.)

[141] Pablo Trinidad, David Benavides, Amador Durán, Antonio Ruiz Cortés, and Miguel Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008. URL: https://doi.org/10.1016/j.jss.2007.10.030. (Zitiert auf Seite 196.)

[142] Mitchell M. Tseng and Jianxin Jiao. *Mass Customization*, S. 684–709. Wiley, 2001. URL: http://dx.doi.org/10.1002/9780470172339.ch25. (Zitiert auf Seite 19.)

[143] G.S. Tseytin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, S. 115—125, 1968. (Zitiert auf Seite 167.)

[144] Malik Ayed Tubaishat, Peng Zhuang, Qi Qi, and Yi Shang. Wireless sensor networks in intelligent transportation systems. *Wireless Communications and Mobile Computing*, 9(3):287–302, 2009. URL: https://doi.org/10.1002/wcm.616. (Zitiert auf Seite 15.)

[145] Shiyong Wang, Jiafu Wan, Daqiang Zhang, Di Li, and Chunhua Zhang. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101:158–168, 2016. URL: https://doi.org/10.1016/j.comnet.2015.12.017. (Zitiert auf Seite 3.)

[146] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. Optimal reconfiguration of dynamic software product lines based on performance-influence models. In Thorsten Berger, Paulo Borba, Goetz Botterweck, Tomi Männistö, David Benavides, Sarah Nadi, Timo Kehrer, Rick Rabiser, Christoph Elsner, and Mukelabai Mukelabai, editors, *Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, S. 98–109. ACM, 2018. URL: https://doi.org/10.1145/3233027.3233030. (Zitiert auf den Seiten 4, 18 und 39.)

[147] Markus Weckesser, Malte Lochau, Michael Ries, and Andy Schürr. Towards complete consistency checks of clafer models. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Feature-Oriented Software Development*, FOSD 2017, S. 11–20, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3141848.3141850. (Zitiert auf Seite 96.)

[148] Markus Weckesser, Malte Lochau, Michael Ries, and Andy Schürr. Mathematical programming for anomaly analysis of clafer models. In Andrzej Wasowski, Richard F. Paige, and Øystein Haugen, editors, *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018*, S. 34–44. ACM, 2018. URL: https://doi.org/10.1145/3239372.3239398. (Zitiert auf den Seiten 58, 183 und 184.)

[149] Markus Weckesser, Malte Lochau, Thomas Schnabel, Björn Richerzhagen, and Andy Schürr. Mind the Gap! Automated Anomaly Detection for Potentially Unbounded Cardinality-based Feature Models. In *19th Int. Conference on Fundamental Approaches to Software Engineering (FASE)*, S. 158–175, 2016. (Zitiert auf den Seiten 13, 28, 29, 31, 32 und 58.)

[150] Markus Weckesser, Malte Lochau, Thomas Schnabel, Björn Richerzhagen, and Andy Schürr. On automated anomaly detection for potentially unbounded cardinality based feature models. In Jan Jürjens and Kurt Schneider, editors, *Software Engineering 2017, Fachtagung des GI-Fachbereichs Softwaretechnik, 21.-24. Februar 2017, Hannover, Deutschland*, volume P-267 of *LNI*, S. 125–126. GI, 2017. URL: https://dl.gi.de/20.500.12116/1303. (Zitiert auf Seite 58.)

[151] Jules White, Douglas C. Schmidt, David Benavides, Pablo Trinidad, and Antonio Ruiz Cortés. Automated diagnosis of product-line configuration errors in

feature models. In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings*, S. 225–234. IEEE Computer Society, 2008. URL: https://doi.org/10.1109/SPLC.2008.16. (Zitiert auf Seite 34.)

[152] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things - A survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015. URL: https://doi.org/10.1007/s10796-014-9489-2. (Zitiert auf Seite 3.)

[153] Matthias Wichtlhuber, Björn Richerzhagen, Julius Rückert, and David Hausheer. TRANSIT: supporting transitions in peer-to-peer live video streaming. In *2014 IFIP Networking Conference, Trondheim, Norway, June 2-4, 2014*, S. 1–9, 2014. URL: https://doi.org/10.1109/IFIPNetworking.2014.6857079. (Zitiert auf Seite 15.)

[154] H Paul Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 2013. (Zitiert auf den Seiten 88 und 162.)

[155] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in C compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, S. 283–294, 2011. URL: https://doi.org/10.1145/1993498.1993532. (Zitiert auf Seite 178.)

[156] Fang Yu, Tevfik Bultan, and Erik Peterson. Automated size analysis for OCL. In Ivica Crnkovic and Antonia Bertolino, editors, *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, S. 331–340. ACM, 2007. URL: https://doi.org/10.1145/1287624.1287671. (Zitiert auf Seite 201.)

[157] Lamia Abo Zaid, Frederic Kleinermann, and Olga De Troyer. Applying semantic web technology to feature modeling. In Sung Y. Shin and Sascha Ossowski, editors, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, S. 1252–1256. ACM, 2009. URL: https://doi.org/10.1145/1529282.1529563. (Zitiert auf Seite 198.)

[158] Teruyoshi Zenmyo, Hideki Yoshida, and Tetsuro Kimura. A self-healing technique based on encapsulated operation knowledge. In *Proceedings of the 3rd International Conference on Autonomic Computing, ICAC 2006, Dublin, Ireland, 13-16 June 2006*, S. 25–32, 2006. URL: http://ieeexplore.ieee.org/document/1662378/. (Zitiert auf Seite 3.)

[159] Wei Zhang, Hua Yan, Haiyan Zhao, and Zhi Jin. A bdd-based approach to verifying clone-enabled feature models' constraints and customization. In Hong Mei, editor, *High Confidence Software Reuse in Large Systems, 10th International Conference on*

Software Reuse, ICSR 2008, Beijing, China, May 25-29, 2008, Proceedings, volume 5030 of *Lecture Notes in Computer Science*, S. 186–199. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68073-4_18. (Zitiert auf den Seiten 8, 34 und 197.)

[160] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Towards a UML profile for software product lines. In Frank van der Linden, editor, *Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers*, volume 3014 of *Lecture Notes in Computer Science*, S. 129–139. Springer, 2003. URL: https://doi.org/10.1007/978-3-540-24667-1_10. (Zitiert auf Seite 48.)

A

ABBILDUNGSVERZEICHNIS

B

TABELLENVERZEICHNIS

C

VERZEICHNIS DER CODE-LISTINGS

D

VERZEICHNIS DER ALGORITHMEN

E

VERZEICHNIS DER BEISPIELE

F

DETAILLIERTES INHALTSVERZEICHNIS